JOURNAL OF THE ROYAL LAUREATES ACADEMY

www.rlaindia.org

INTELLIGENT CACHE RESOURCE MANAGEMENT IN MULTICORE PROCESSORS: MACHINE LEARNING–DRIVEN PARTITIONING, ADAPTATION, AND PERFORMANCE OPTIMIZATION

Sajjade Zeba Shazesh

Research Scholar, Jayoti Vidyapeeth Women's University, Jaipur, Rajasthan

Dr. Sushma Agarwal

Research Supervisor, Jayoti Vidyapeeth Women's University, Jaipur, Rajasthan

ABSTRACT

Shared last-level caches in multicore processors are critical performance resources because multiple applications and processor cores compete for limited cache capacity. Conventional static or heuristic cache partitioning methods may perform poorly when workload behavior changes dynamically, particularly when applications exhibit different levels of cache sensitivity and phase behavior. Machine learning offers an attractive approach for predicting cache demand, identifying workload characteristics, and dynamically reallocating shared cache resources. This paper examines machine learning–driven cache resource management for multicore processors, with particular emphasis on cache partitioning, workload classification, adaptive allocation, interference control, and performance optimization. The study discusses features derived from hardware performance monitoring, learning-based prediction, dynamic cache allocation, fairness-aware management, and coordination between cache capacity and processor resources. Existing research indicates that intelligent cache management can improve throughput, fairness, and energy efficiency compared with static allocation, while recent approaches demonstrate the value of predicting changing application requirements before reallocating cache resources.

Keywords: Multicore Processor, Cache Partitioning, Last-Level Cache, Machine Learning, Resource Management, Cache Allocation, Performance Optimization, Workload Classification, Quality of Service, Hardware Performance Counters.

I. INTRODUCTION

The continuing increase in the number of processor cores has transformed the organization of modern computing systems. Multicore processors can execute multiple application threads concurrently, but they also introduce competition for shared hardware resources. Among these resources, the shared last-level cache plays a particularly important role because it provides faster access than main memory and is shared by multiple cores and workloads. Efficient management of this limited capacity can substantially influence application performance, fairness, memory traffic, and energy consumption. Cache partitioning divides shared cache capacity among applications or cores to control interference and allocate capacity according to workload requirements. However, static partitioning may become inefficient when workloads change their behavior during execution. Research surveys have identified fairness, quality-of-service guarantees, cache sensitivity, and performance isolation as major challenges in shared-cache management.

The difficulty is intensified by workload diversity. Compute-intensive applications may receive little benefit from additional cache capacity, whereas memory-intensive applications may be highly sensitive to cache allocation. Two applications can also have different working-set sizes and may interfere with one another in very different ways. Furthermore, applications often exhibit phase behavior, meaning that their cache requirements change over time. A fixed cache allocation selected at program launch may therefore become unsuitable during later execution. Recent work on dynamic cache apportioning has explicitly identified fine-grained phase behavior as a reason why conventional partitioning mechanisms can react too slowly to changing cache requirements.

Machine learning provides an opportunity to address this dynamic management problem through workload characterization and prediction. Instead of relying solely on fixed heuristics, a learning-based cache manager can observe hardware performance counters, cache misses,

memory-level parallelism, instructions per cycle, bandwidth consumption, and other runtime indicators, then predict how an application is likely to respond to different cache allocations. Earlier research established the broader feasibility of machine learning for coordinated resource management in chip multiprocessors, while later cache-specific work has applied machine learning to dynamic cache apportioning.

One notable example is Com-CAS, which combines compiler-derived information with machine learning to predict cache requirements and dynamically apportion the last-level cache using Intel Cache Allocation Technology. The reported evaluation found improvements in average weighted throughput over an unpartitioned system and over an earlier partitioning approach, while also considering application-level service constraints. Other recent systems have approached the problem through lightweight runtime classification. LFOC+, for example, uses performance-monitoring data to classify applications according to cache sensitivity and contentiousness and then performs cache clustering to improve fairness on commodity multicore systems.

II. MACHINE LEARNING–DRIVEN CACHE PARTITIONING AND ADAPTIVE RESOURCE ALLOCATION

The fundamental task of intelligent cache management is to determine how the available shared cache should be divided among competing applications. In a static partitioning approach, the cache allocation remains fixed for a significant period. Such a strategy is straightforward and imposes little runtime overhead, but it assumes that workload requirements remain relatively stable. This assumption is often unrealistic in heterogeneous multicore systems. Applications can alternate between compute-intensive and memory-intensive phases, change their working-set size, or interact differently with co-running workloads. Consequently, the cache allocation that is optimal at one point may become inefficient later.

A machine learning–driven system introduces an observation-and-decision loop. First, the processor collects runtime information. Useful features may include last-level cache misses, cache references, instructions retired, cycles, branch behavior, memory bandwidth, miss intensity, instructions per cycle, and estimates of memory-level parallelism. Hardware performance monitoring units provide much of this information with comparatively low

overhead. The feature vector can then be processed by a prediction model that estimates cache sensitivity, expected performance under alternative allocations, or application classes.

Workload classification is a particularly useful first step. Applications can be categorized as cache-sensitive, cache-insensitive, bandwidth-intensive, compute-intensive, or interference-sensitive. A learning model can infer these categories from runtime behavior rather than requiring manual application profiling. Supervised learning can be trained on previously collected workload data, whereas clustering approaches can discover groups of applications with similar behavior without requiring labeled training sets. Online learning can further adapt the model as new workload patterns emerge.

A machine learning model can approximate the relationship between cache allocation and performance. This is valuable because the exact performance landscape may be difficult to model analytically for arbitrary applications. Traditional optimization methods often require exploring multiple cache configurations, which can create management latency. Recent work on online coordinated cache and memory-bandwidth management identifies this configuration-search cost as a significant challenge for dynamic resource managers. A predictive model can reduce the number of configurations that need to be tested directly by narrowing the search space before allocation changes are made.

One useful strategy is to predict the performance curve of each application with respect to cache allocation. The system can estimate the expected speedup obtained from each additional cache partition and identify diminishing returns. The manager can then allocate cache to applications according to marginal performance benefit. Such an approach is more efficient than simply assigning equal partitions or using application priority alone.

Machine learning can also support dynamic cache partitioning at finer time scales. Application behavior may change within milliseconds or even shorter intervals. A learning model can recognize phase transitions by monitoring abrupt changes in performance-counter patterns. When a phase transition is detected, the manager can reevaluate the cache allocation. This proactive strategy is potentially superior to waiting until performance has already degraded. Com-CAS represents an example of this broader concept by combining predictive learning with

dynamic cache apportionment. Its reported evaluation demonstrated improvements in weighted throughput relative to unpartitioned and earlier partitioning systems.

However, prediction accuracy is not the only concern. A model with high computational cost may negate the benefits of intelligent management. Therefore, practical cache managers require lightweight inference. Decision trees, linear models, small neural networks, regression models, and clustering methods may be preferable to very large models because inference can be executed quickly and with modest energy consumption. The model may also operate at a lower frequency than the processor pipeline, periodically updating decisions rather than making decisions for every memory access.

Hardware support for cache partitioning is another important consideration. Technologies such as Intel Cache Allocation Technology allow software to control cache capacity assigned to classes of service. Such mechanisms make dynamic partitioning more practical because machine-learning decisions can be translated into hardware allocation policies without redesigning the processor cache. Research on real-time systems has also demonstrated that hardware-supported cache partitioning can improve predictability by isolating critical tasks from cache contention.

Cache partitioning can further be combined with cache clustering. Rather than assigning a completely isolated partition to every application, compatible workloads can share partitions. This approach can improve resource utilization when applications have complementary behavior. LFOC+ is an example of a runtime cache-clustering strategy that uses performance-monitoring information to classify applications by cache sensitivity and contentiousness, with reported improvements in fairness relative to earlier policies.

III. PERFORMANCE OPTIMIZATION, SYSTEM INTEGRATION, AND RESEARCH CHALLENGES

The effectiveness of intelligent cache management should be evaluated using a broad set of performance metrics. Throughput is often the primary system-level objective because efficient cache allocation can reduce memory stalls and improve processor utilization. Weighted speedup is commonly used to evaluate aggregate performance while accounting for the performance of

individual applications. However, throughput alone may conceal serious unfairness. For multi-tenant or quality-of-service-sensitive environments, maximum slowdown, minimum performance guarantee, response-time variation, and fairness indexes should also be incorporated.

The interaction between cache and memory bandwidth is especially significant. An application that receives insufficient cache may generate many memory requests and consume substantial bandwidth. Conversely, assigning more cache may reduce memory traffic and benefit both the application and other cores. Recent work on coordinated last-level-cache and memory-bandwidth management emphasizes that these resources should sometimes be considered jointly rather than independently. A machine-learning model can learn these interactions from performance-counter data and make coordinated decisions.

Memory-level parallelism is another important factor. An application with substantial memory-level parallelism may hide some cache-miss latency and therefore respond differently to additional cache capacity than an application with limited parallelism. Research on coordinated processor configuration and cache partitioning has highlighted memory-level parallelism as an important variable when predicting the effect of resource allocation on performance. This suggests that future intelligent cache managers should not rely exclusively on cache-miss rates but should incorporate a broader representation of application execution behavior.

The timing of resource reallocation also strongly affects performance. Frequent partition changes can create instability and management overhead. Cache contents may be displaced when partition sizes change, temporarily reducing locality. Conversely, reallocating too slowly can cause the manager to miss opportunities created by workload phase changes. An adaptive policy should therefore balance responsiveness against stability. One approach is to use a two-level control strategy in which a lightweight detector identifies significant phase changes while a more computationally intensive optimization model is invoked only after a change is detected.

Model training represents another challenge. A supervised model requires representative training data covering a large variety of applications and co-running combinations. A model trained on one workload set may not generalize well to another system, operating system, cache

organization, or application domain. Transfer learning and online adaptation could reduce this problem. The model can begin with a pre-trained representation and update selected parameters using new runtime observations.

The problem of concept drift is particularly important. Application behavior may change over time, and system conditions may also vary because of changes in memory pressure, processor frequency, or co-running workloads. A model that was accurate previously may gradually become inaccurate. Online learning and periodic retraining can address this issue, but they introduce additional computational overhead. Practical designs may therefore use confidence estimation to determine when model predictions are sufficiently reliable and when new data should be collected.

Explainability is another consideration. Hardware resource managers are often expected to operate predictably, especially in real-time systems. A black-box model that suddenly changes cache allocation without an interpretable reason may be difficult to validate. Simpler models or hybrid approaches can improve transparency. A rule-based controller can handle safety constraints, while a machine learning predictor supplies performance estimates. Such a hybrid architecture combines the adaptability of learning with the predictability of conventional control logic.

Security and isolation also matter in shared-cache systems. Cache behavior can reveal information about co-running applications, creating potential side-channel risks. Partitioning can reduce some forms of interference by isolating workloads, but dynamic reassignment must not inadvertently weaken security boundaries. Intelligent resource management must therefore include explicit isolation policies and, where necessary, reserved secure cache regions.

Hardware implementation is a further challenge. A learning-based resource manager requires performance monitoring, feature collection, inference logic, decision control, and communication with the cache-allocation mechanism. Excessive hardware complexity could offset performance benefits. A practical implementation should minimize critical-path impact and execute the learning model on a management processor, operating-system service, or low-

frequency controller rather than directly in the main pipeline. This separation enables intelligent optimization without affecting instruction execution timing.

Experimental evaluation should ideally include benchmarks with diverse behavior, multiple core counts, varying cache sizes, and different memory systems. Benchmarking should compare the learning-based approach against unpartitioned caching, static equal partitioning, heuristic partitioning, utility-based partitioning, cache clustering, and other state-of-the-art policies. Metrics should include weighted speedup, throughput, fairness, maximum slowdown, energy, cache misses, memory bandwidth, and management overhead.

Real-system evaluation is especially important. Simulation can provide extensive design flexibility, but simulator results do not always predict hardware behavior accurately. Recent cache-clustering research demonstrates the feasibility of implementing runtime cache management directly in a Linux environment on commodity multicore hardware, which is an important step toward practical deployment. Similarly, hardware-supported cache allocation technologies provide a direct mechanism through which operating systems can enforce learned resource allocations.

Several research directions appear promising. First, reinforcement learning could be used for sequential cache-allocation decisions. Rather than predicting performance directly, an agent could learn a policy that chooses allocations to maximize long-term utility. The main challenge is ensuring exploration does not harm applications during training. Safe reinforcement learning and constrained policy optimization may provide solutions.

Second, multi-resource learning could jointly manage cache capacity, memory bandwidth, processor frequency, and core assignment. Earlier research has demonstrated that coordinated resource management can produce energy benefits that are difficult to obtain when each resource is optimized independently. Machine learning could provide a unified prediction framework for these interacting resources.

IV. CONCLUSION

Intelligent cache resource management represents an important direction for improving the efficiency of multicore processors as the number of cores, workload diversity, and demand for shared-memory performance continue to increase. The shared last-level cache is a finite and highly valuable resource, and inefficient allocation can produce cache contention, memory-bandwidth pressure, unfairness, and unpredictable application performance. Static partitioning policies are attractive because of their simplicity, but they cannot adequately respond to applications whose cache demands change across execution phases. The central contribution of machine learning is the ability to recognize these changing patterns, estimate future cache requirements, and support more adaptive resource-allocation decisions.

The research examined in this paper demonstrates the feasibility of combining runtime observations, machine learning, and hardware-supported cache partitioning. Cache-management surveys have established that intelligent allocation is necessary for achieving performance isolation and quality-of-service objectives in shared-cache multicore systems. More recent approaches demonstrate that machine learning can be used to predict cache requirements and perform dynamic allocation at finer temporal granularities. The Com-CAS approach illustrates the potential of compiler-informed learning and proactive cache apportionment, with reported improvements in weighted throughput relative to competing approaches. Runtime cache-clustering approaches such as LFOC+ further show that performance-monitoring information can support practical workload classification and fairness-aware resource organization on commodity multicore processors.

A key conclusion is that intelligent cache management should optimize more than cache hit rate. A high-performing multicore system must balance throughput, fairness, latency, energy, memory bandwidth, and quality of service. Cache capacity interacts strongly with processor frequency, memory-level parallelism, and bandwidth consumption. Research on coordinated resource management indicates that jointly considering cache and processor resources can yield substantial energy benefits under performance constraints. Recent work on online cache and memory-bandwidth management also highlights the importance of reducing configuration-search overhead when dynamic resource allocation is performed without prior workload knowledge.

The practical adoption of machine learning therefore depends on low-overhead implementation. A learning model that consumes excessive computation or requires frequent exploratory measurements may reduce the performance benefits obtained from better cache allocation. Lightweight predictors, online adaptation, hybrid rule-learning controllers, and confidence-aware decision mechanisms are attractive because they preserve responsiveness while limiting management cost. Hardware support such as Intel Cache Allocation Technology makes it possible for software policies to translate predicted allocations into enforceable cache partitions, and related research has shown that hardware-assisted partitioning can improve predictability for real-time workloads.

Fairness and security must remain integral to intelligent cache management. A system optimized exclusively for aggregate throughput may severely degrade an individual application, particularly in multi-tenant environments. Similarly, dynamic allocation mechanisms should maintain security and isolation requirements when applications have different trust levels. The learning objective must therefore incorporate explicit service constraints and isolation policies rather than treating performance maximization as the only goal.

V. REFERENCES

1. Mittal, S. (2017). A survey of techniques for cache partitioning in multicore processors. *ACM Computing Surveys*, *50*(2), Article 27, 1–39.

2. Bitirgen, R., Ipek, E., & Martinez, J. F. (2008). Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach. In *Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture* (pp. 318–329).

3. Chatterjee, B., Khan, S., & Pande, S. (2021). Effective cache apportioning for performance isolation under compiler guidance. *Proceedings of the International Conference on Parallel Processing*.

4. Nejat, M., Manivannan, M., Pericas, M., & Stenstrom, P. (2019). Coordinated management of processor configuration and cache partitioning to optimize energy under QoS constraints. *arXiv*.

5. Saez, J. C., Castro, F., Fanizzi, G., & Prieto-Matias, M. (2024). LFOC+: A fair OS-level cache-clustering policy for commodity multicore systems. *arXiv*.

6. Balakrishnan, P., Rajesh, M., & Rajesh, R. (2025). Partitioned cache aware dynamic scheduling for real-time applications on multicore processors. *Defence Science Journal*.

7. Hassidim, A., Kaplan, H., & Tuval, O. (2012). Joint cache partition and job assignment on multi-core processors. *arXiv*.

8. Wang, Z., et al. (2016). Reconfigurable cache for real-time MPSoCs: Scheduling and implementation. *Microprocessors and Microsystems, 42*, 200–214.

9. *PaLLOC: Pairwise-based low-latency online coordinated resource manager of last-level cache and memory bandwidth on multicore systems*. (2025). *Journal of Systems Architecture*, *164*, 103427.

10. Beckmann, N., & Sanchez, D. (2016). Modeling cache performance beyond LRU. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA).*