Reinforcement Learning-Driven Adaptive Prefetch Aggressiveness Control for Enhanced Performance in Parallel System Architectures

Huijing Yang, *Student Member, IEEE*, Juan Fang, *Senior Member, IEEE*, Yumin Hou, Xing Su, and Neal N. Xiong, *Senior Member, IEEE*

***Abstract*—In modern parallel system architectures, prefetchers are essential to mitigating the performance challenges posed by long memory access latencies. These architectures rely heavily on efficient memory access patterns to maximize system throughput and resource utilization. Prefetch aggressiveness is a central parameter in managing these access patterns; although increased prefetch aggressiveness can enhance performance for certain applications, it often risks causing cache pollution and bandwidth contention, leading to significant performance degradation in other workloads. While many existing prefetchers rely on static or simple built-in aggressiveness controllers, a more flexible, adaptive approach based on system-level feedback is essential to achieving optimal performance across parallel computing environments. In this paper, we introduce an Adaptive Prefetch Aggressiveness Control (APAC) framework that leverages Reinforcement Learning (RL) to dynamically manage prefetch aggressiveness in parallel system architectures. The APAC controller operates as an RL agent, which optimizes prefetch aggressiveness by dynamically responding to system feedback on prefetch accuracy, timeliness, and cache pollution. The agent receives a reward signal that reflects the impact of each adjustment on both performance and memory bandwidth, learning to adapt its control strategy based on workload characteristics. This data-driven adaptability makes APAC particularly well-suited for parallel architectures, where efficient resource management across cores is essential to scaling system performance. Our evaluation with the ChampSim simulator demonstrates that APAC effectively adapts to diverse workloads and system configurations, achieving performance gains of 6.73% in multi-core systems compared to traditional Feedback Directed Prefetching (FDP). By improving memory bandwidth utilization, reducing cache pollution, and minimizing inter-core interference, APAC significantly enhances prefetching performance in multi-core processors. These results underscore APAC's potential as a robust solution for performance optimization in parallel system architectures, where efficient resource management is paramount for scaling modern processing environments.**

Received 12 November 2024; revised 12 February 2025; accepted 9 March 2025. Date of publication 12 March 2025; date of current version 7 April 2025. This work was supported by National Natural Science Foundation of China under Grant 62276011 and Grant 61202076, and in part by Beijing Municipal Natural Science Foundation under Grant 4192007. Recommended for acceptance by A. Li. *(Corresponding author: Juan Fang.)*

Huijing Yang, Juan Fang, Yumin Hou, and Xing Su are with the College of Computer Science, Beijing University of Technology, Beijing 100124, China (e-mail: yanghj1998@outlook.com; fangjuan@bjut.edu.cn; houyumin@bjut.edu.cn; xingsu@bjut.edu.cn).

Neal N. Xiong is with the Department of Computer, Mathematical and Physical Sciences, Sul Ross State University, Alpine, TX 79830 USA (e-mail: xiongnaixue@gmail.com).

Digital Object Identifier 10.1109/TPDS.2025.3550531

***Index Terms*—Data prefetchers, memory bandwidth, prefetcher aggressiveness controller, parallel system architectures, reinforcement learning.**

I. Introduction

MODERN multi-core systems leverage hardware prefetchers to alleviate the impact of long memory access latencies, thereby significantly enhancing performance in parallel computing environments [1], [2], [3]. In these parallel system architectures, characterized by multiple processing elements operating concurrently, efficient prefetching is critical to maintaining high throughput and reducing memory-related bottlenecks. Prefetcher aggressiveness, a crucial tuning parameter, dictates the degree to which the prefetcher speculatively retrieves cache lines from deeper levels of the memory hierarchy. This parameter directly influences system performance by controlling the balance between reducing memory access latency and mitigating potential negative impacts such as bandwidth overuse and cache pollution. While increasing prefetcher aggressiveness can improve system performance, particularly in parallel system architectures, it may also introduce bandwidth contention, cache pollution, and performance degradation in some workloads. These issues become more pronounced in parallel system architectures, where shared memory resources, such as bandwidth and caches, are heavily contested [4], [5], [6]. As depicted in Fig. 1, modern processors typically incorporate a three-level on-chip cache hierarchy. Prefetching is initiated by L2 cache accesses, and prefetch requests can be fulfilled either in the L2 cache or the Last Level Cache (LLC). Prefetching can improve performance by predicting and issuing memory accesses ahead of time, thus reducing demand misses and alleviating memory bottlenecks [7], [8]. However, aggressive prefetching increases DRAM traffic, which can lead to memory bandwidth contention and latency penalties, particularly in parallel computing environments where multiple applications and processes contend for shared resources. Excessive prefetching also leads to cache pollution, unnecessary memory accesses, and higher energy consumption [9], [10], [11], [12]. In parallel system architectures with multiple cores, these negative impacts exacerbate the challenges in resource coordination, especially for bandwidth-intensive workloads [13], [14], [15], [16].

Recent prefetching techniques have demonstrated performance gains in single-core architectures but often increase

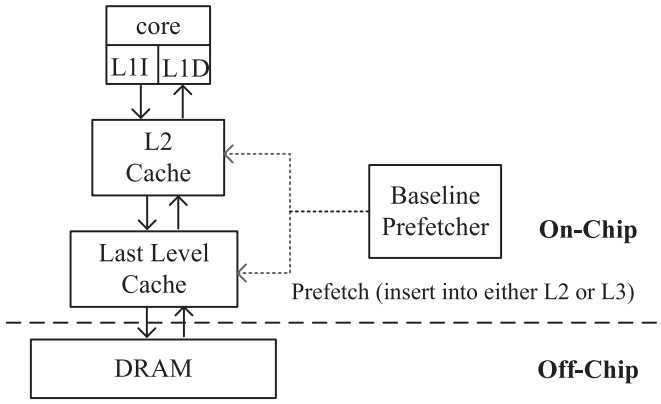


Fig. 1. The hierarchy of the memory subsystem.

DRAM traffic. For instance, Bingo spatial prefetcher [2] improves coverage by predicting memory footprints, but overpredictions can waste memory bandwidth. Other techniques, such as the Zero Pollution Prefetcher (ZPP) [17], aim to reduce cache pollution by storing prefetched blocks in the LLC's stale block locations. These methods, however, fail to effectively manage shared system resources, such as memory bandwidth, particularly in parallel system architectures. Traditional prefetchers often utilize static aggressiveness controllers or rule-based adjustments, which do not account for dynamic workload changes or system-level feedback in multi-core settings [8], [11], [17], [18], [19]. Consequently, these methods often underperform in bandwidth-constrained scenarios where adaptive resource management is crucial.

More recent prefetchers aim to forecast irregular access patterns by utilizing a sequence of address deltas or combining access history and the Program Counter (PC) [20], [18], [19], [21]. For example, VLDP [18], Berti [22], and IPCP [23] focus on deltas between accessed addresses to predict subsequent address deltas. These prefetchers typically employ conservative built-in prefetch throttling mechanisms, where prefetch commands are issued to the processor based on accurate predictions above a threshold. However, while effective in certain scenarios, traditional rule-based methods used to adjust prefetcher aggressiveness dynamically through feedback information [8], [11] exhibit limitations in multi-core systems.

In parallel system architectures, several key challenges exist in the design of RL-based prefetch aggressiveness controllers, particularly in bandwidth-constrained and scalable systems: (1) RL methods require significant storage and computation to maintain state-action mappings, which can strain hardware resources; (2) traditional RL methods need multiple interactions to converge to optimal strategies, hindering real-time decision-making in dynamic environments; and (3) excessive prefetching by one core in multi-core systems can degrade the performance of others due to shared bandwidth.

To address these challenges, this paper proposes a Reinforcement Learning-based Adaptive Prefetch Aggressiveness Control Mechanism (APAC) for parallel system architectures, as illustrated in Fig. 2. APAC leverages RL to dynamically adjust prefetch aggressiveness, while optimizing issues related

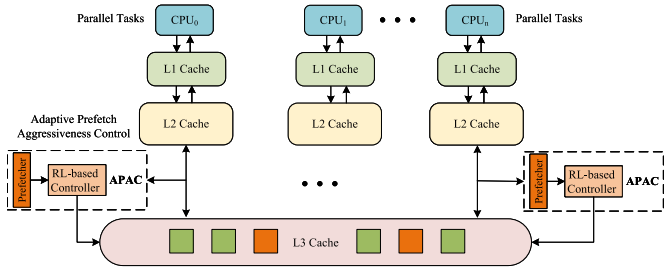


Fig. 2. The APAC mechanism for adaptive prefetch aggressiveness control in parallel system architectures.

to storage and computational overhead, real-time adaptability, and resource contention in parallel systems. The contributions of this paper can be summarized as follows.

1) APAC reduces storage requirements and computational latency by using compressed state representations and a low-overhead reward mechanism, maintaining efficient decision-making in hardware-limited environments.
2) APAC accelerates the learning process and reduces exploration time by dynamically adjusting the reward function and incorporating system-level feedback, enabling fast and effective prefetch aggressiveness adjustments.
3) APAC balances prefetch aggressiveness across cores in multi-core systems through a joint optimization strategy, monitoring bandwidth utilization to coordinate resource usage and prevent performance degradation from overprefetching by a single core.
4) Extensive experiments show that APAC performs better than the current prefetcher aggressiveness controllers in various workloads and system configurations. It can effectively improve system performance and bandwidth efficiency.

The rest of the paper is organized as follows. Section II introduces the related work on prefetchers. We present the preliminaries in Section III. In Section IV, we describe the design philosophy and system architecture of our proposed APAC framework. We present the experimental evaluation and results in Section V. Section VI discusses the impact of varying experimental settings on the results. Finally, Section VII concludes with remarks on future work.

II. Related Work

A. Prefetchers With Built-In Prefetch Throttling Mechanism

In parallel system architectures, efficient prefetching becomes more challenging due to shared resource contention among multiple cores and processing units. AlBarakat et al. propose SLAP-CC [24], a compressibility-aware prefetching technique that adapts prefetch aggressiveness based on workload compressibility to maximize prefetch coverage. However, such techniques often struggle in parallel systems where memory requests from multiple cores introduce additional complexity and demand adaptive prefetching. SPP [25] generates a signature for each delta offset pattern, assessing the probability of delta predictions corresponding to each signature and employing a throttling mechanism that adjusts prefetching lookahead

based on accuracy. Instruction Pointer Classifier-based spatial hardware Prefetching (IPCP) [23] employs instruction pointer classification for spatial prefetching at the L1 cache. It combines three sub-prefetchers and adjusts the prefetch degrees based on prefetch accuracy. In addition to IPCP, Classifying Memory Access Patterns for Prefetching [20] presents a software-based methodology that classifies memory access patterns, such as strides and reuse, to guide L1 cache prefetching. It analyzes instruction dataflow to identify these patterns and applies them for prefetch decision-making, offering scalability and the ability to tune prefetcher parameters offline without exhaustive configuration searches. These state-of-the-art techniques demonstrate significant improvements in single-core environments; however, in parallel architectures, the increased DRAM traffic from additional prefetch requests (22.0% and 23.8% for IPCP and SPP, respectively) can strain shared bandwidth, limiting their effectiveness as DRAM contention rises in multi-core settings.

B. Prefetcher Aggressiveness Controllers

Given the increased need to manage shared resources in parallel system architectures, specialized prefetcher aggressiveness controllers have been developed to enhance prefetch efficiency. These controllers adaptively set the prefetch degree and distance based on feedback metrics such as prefetch accuracy, cache pollution, and DRAM bandwidth. FDP [11] uses a rule-based controller to dynamically adjust prefetcher aggressiveness, aiming to balance the benefits of prefetching with the risk of cache pollution. Unlike FDP, which operates at the individual prefetcher level and adapts based on predefined heuristics, HPAC [26] focuses on mitigating inter-core interference by globally coordinating prefetching behavior. It prioritizes reducing contention for shared resources through system-wide bandwidth management and interference mitigation, rather than optimizing prefetching based on per-core performance feedback. Similarly, the Synergistic Prefetcher Aggressiveness Controller (SPAC) [13] introduces throttling decisions to manage multiple prefetchers for fair speedup across cores in a shared-memory environment.

DeepP [27] uses deep learning for dynamic prefetch adjustment on IBM POWER8 processors, improving parallel system performance through workload-aware tuning. CLIP [6] is designed for optimizing L1 cache prefetching by identifying and prefetching critical load addresses that have a high probability of cache hits. It uses fine-grained prefetch accuracy, targeting critical load addresses that can be prefetched with high accuracy. CLIP's design is tightly integrated with the Berti prefetcher [22], a first-level data cache prefetcher that selects the best local deltas based on demand accesses issued by the same instruction. Berti focuses on optimizing prefetch accuracy at the L1 cache level through a high-confidence mechanism that precisely detects timely local deltas with high coverage. Perceptron-Based Prefetch Filtering (PPF) [28] aims to increase prefetch coverage while maintaining accuracy by filtering out inaccurate L2 prefetch predictions. Another approach [29] introduces the pure prefetch coverage metric to more accurately assess prefetch effectiveness amid concurrent memory accesses, offering an adaptation layer suited for parallel processing environments. Near-side prefetch throttling [30] further tailors prefetch distance based on late prefetch rates, keeping distances conservative unless needed.

Despite their contributions, these controllers often rely on manually tuned parameters, which may not adapt well to different architectures or workload variations typical in parallel systems. Furthermore, these controllers generally do not account for the cumulative, long-term impact of aggressive prefetching in multi-core setups where shared resources and inter-core interference can vary dynamically. As such, existing controllers often lose their advantage across a range of configurations, limiting their scalability and adaptability in complex parallel architectures.

C. Reinforcement Learning in Computer Architecture

Machine learning-based techniques have increasingly been applied to architectural tasks, including cache management [31], [32], branch prediction [33], [34], and hardware prefetching [28], [35], [36]. For example, PATHFINDER [37] introduces a neural-inspired LLC prefetcher that leverages spiking neural networks to learn delta patterns in real-time. This approach emphasizes dynamic prefetching at the LLC level by adapting to temporal memory access patterns, allowing it to make real-time adjustments. Similarly, Voyager [38] is a hierarchical neural model designed for irregular address pattern prefetching at the LLC level. Voyager aims to address complex memory access patterns by using a hierarchical approach, optimizing LLC-level prefetching. Both PATHFINDER and Voyager operate at the LLC level, which makes them orthogonal to APAC's emphasis on L2 cache prefetch aggressiveness control. These approaches, while effective in enhancing prediction accuracy, encounter two main challenges in parallel system architectures. First, the model sizes often exceed Last Level Cache (LLC) capacity, constraining their deployment in shared-memory architectures. Second, the computational load of large models introduces delays that hinder real-time responsiveness essential for parallel system architectures.

Reinforcement learning presents a solution to some of these limitations by focusing on real-time, environment-driven learning that does not require pre-existing datasets. In parallel system architectures, RL-based approaches allow dynamic and context-specific decisions, where optimal actions are continuously refined based on resource utilization feedback and system performance metrics. This adaptability makes RL particularly suitable for prefetching in parallel system architectures, as it can efficiently navigate the complexity of shared resource contention without prior knowledge, dynamically adjusting actions based on feedback from ongoing processes [39], [40], [41], [42], [43], [44].

Pythia [44] is an RL-based prefetching approach that optimizes the prefetch offset, which refers to the address difference between the predicted and requested cache lines. Using program context information, such as program counters and cache addresses, Pythia predicts the optimal offset, improving data fetch accuracy and reducing access latency. Unlike Pythia,

which focuses on selecting the prefetch offset, the goal of our proposed APAC is to dynamically adjust the prefetch degree of the base prefetcher, which refers to the number of cache blocks fetched in each operation. APAC uses RL to dynamically control the prefetch degree based on system performance feedback, ensuring efficient resource utilization.

RL-based methods have shown promise in addressing the long-term impacts of prefetcher aggressiveness by learning policies that adaptively manage resource usage across diverse workloads, an essential characteristic in scalable parallel architectures where multiple cores and threads may simultaneously compete for memory and bandwidth. However, challenges remain in optimizing RL model overhead and ensuring it scales effectively within the hardware constraints of parallel systems.

III. Preliminaries

In this section, we first briefly describe the basic principles of reinforcement learning (RL). Then, we explain why RL can effectively serve as a foundation for a prefetcher aggressiveness controller to improve system performance and bandwidth efficiency, especially in parallel system architectures.

A. Reinforcement Learning

RL seeks to develop a policy to maximize total rewards through an RL agent's interaction with its environment. In dynamic environments like parallel systems, the need for real-time adaptation is critical, as system conditions can change rapidly. For this reason, we selected the SARSA (State-Action-Reward-State-Action) algorithm over other RL methods, such as Deep Q-Networks (DQN). While DQN employs deep neural networks that are computationally intensive and require significant training, SARSA is a simpler, model-free algorithm that updates policies in real-time based on immediate feedback. This makes SARSA especially suitable for prefetcher aggressiveness control, where fast adaptation to workload changes and minimal computational overhead are essential.

As a type of dynamic programming, RL is computationally efficient and requires minimal memory, attributes that align well with parallel system architectures, where resource efficiency is critical [45], [46]. The agent interacts with the environment in discrete timesteps. At each timestep t, the agent observes a state S_t, takes an action A_t, and experiences a transition, leading to a new state S_{t+1} and an immediate reward R_{t+1}. The RL process, guided by a reward scheme, seeks to determine optimal actions to maximize cumulative rewards, adapting flexibly to the complex and dynamic workloads typical of parallel systems.

The agent's policy defines the action to be taken in each state. The objective of the agent is to find an optimal policy to maximize its long-term cumulative reward. Q-value ($Q(S, A)$) represents the expected long-term cumulative reward associated with taking action A in state S, capturing both immediate and future gains, which is especially valuable in parallel architectures where delayed impacts of decisions often influence overall performance.

1) Q-Value Updating: The SARSA algorithm is used to iteratively optimize the Q-value of the previous state-action pair $Q(S_t, A_t)$, as shown in (1) :

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma Q\left(S_{t+1}, A_{t+1}\right) - Q\left(S_t, A_t\right)\right]. \quad (1)$$

where α controls how quickly the agent incorporates new information, influencing the update magnitude of model parameters. Meanwhile, γ plays a crucial role in striking a balance between immediate and future rewards.

2) Optimizing Policy: Finding the optimal policy entails a delicate balance between exploration and exploitation to maximize long-term cumulative rewards. The ϵ-greedy agent introduces randomness in action selection, governed by the probability ϵ, facilitating both exploration and exploitation for effective learning.

B. RL's Applicability to Prefetcher Aggressiveness Controlling

In recent research, the RL framework has proven effective for various computer architecture challenges, and we argue that RL is inherently suited to prefetcher aggressiveness control in parallel systems for three primary reasons.

1) Adaptive Learning in Complex State Spaces: In parallel architectures, the high-dimensional state space includes factors like workload type, core utilization, and DRAM bandwidth, all of which vary dynamically. An RL-based prefetcher aggressiveness controller adapts in real-time, optimizing bandwidth efficiency and reducing resource contention by adjusting its policy based on ongoing performance feedback.

2) Online Learning: RL agents learn continuously without needing offline training, making them ideal for parallel systems. This dynamic adaptation allows the prefetcher controller to respond effectively to diverse workloads and system configurations, maintaining high performance across all cores.

3) Ease of Implementation: Complex ML models can be challenging to deploy in parallel systems due to high overhead. RL-based methods, however, are more lightweight and can adapt dynamically, offering a practical and scalable solution for real-time prefetch control with minimal resource demands.

IV. Our Proposed APAC System

In this paper, we introduce APAC, an adaptive aggressiveness controller powered by reinforcement learning, designed to dynamically regulate prefetcher aggressiveness based on real-time system feedback. Tailored for parallel system architectures, APAC addresses the challenges of multi-core environments where multiple cores compete for shared resources, such as memory bandwidth and cache space. By adjusting the prefetch degree, APAC aims to enhance performance across a range of bandwidth-intensive workloads and configurations. This adaptability makes APAC well-suited for complex parallel systems characterized by interdependent cores and threads.

Fig. 3 provides a high-level overview of the proposed RL-based prefetcher aggressiveness controller. APAC operates over one-million-instruction timesteps, during which a collector gathers multiple prefetch effectiveness metrics, such as cache pollution, accuracy, and bandwidth usage, from a baseline prefetcher.

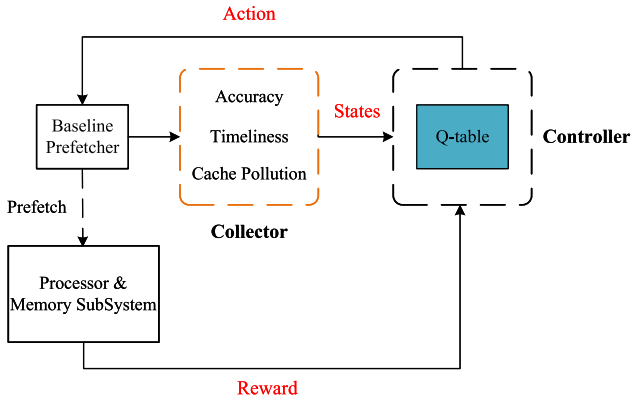


Fig. 3. High-level overview of the RL-based prefetcher aggressiveness controller.

These metrics provide critical feedback on how aggressively prefetching should be adjusted to optimize for performance and bandwidth efficiency, particularly in parallel environments where efficient resource sharing is paramount.

The RL-based controller establishes a state-action pair for each potential adjustment and selects the action associated with the highest Q-value, signaling the adjustment with the most favorable long-term benefit. APAC's address prediction mechanism is founded on a stride-based model that tracks recurring access patterns; prefetched cache lines are stored in the L2 cache to enhance data availability while balancing cache pollution. Although APAC is designed on a stride-based prefetcher, it can easily adapt to other address prediction models, such as sequential or threaded models, offering flexibility in accommodating different prefetching approaches required by various workload demands in multi-core systems.

The reward mechanism of APAC aligns with parallel systems' needs by emphasizing both performance and bandwidth impact, incentivizing adjustments that improve cache utilization without excessive resource consumption. These rewards are used to update Q-values through the SARSA algorithm (detailed in Section III-A1), which enables APAC to learn from each adjustment and refine its control policy over time. By tuning the prefetcher aggressiveness in response to real-time system feedback, APAC ensures more efficient resource utilization in parallel system architectures, thus enhancing overall performance stability across cores and workloads.

A. Formulation of the APAC

In APAC, the prefetcher aggressiveness controller is modeled as an RL-agent, with defined states, actions, and a tailored reward scheme aimed at optimizing performance within parallel system architectures.

1) State: We define the state $S_t = (A_t, L_t, P_t) \in \mathbb{R}^3$ to capture the dynamic characteristics of cache states during time period t. This state includes three core metrics critical in parallel systems: prefetch accuracy A_t, prefetch timeliness L_t, and cache pollution P_t introduced by prefetching activities. These metrics collectively capture how prefetch behavior affects resource sharing and data accessibility in multi-core architectures, where cache pollution and timing are particularly impactful due to the concurrent data access patterns across cores.

1) Prefetch Accuracy A_t: Measures the precision with which the prefetcher predicts future memory accesses, reducing unnecessary memory traffic. High prefetch accuracy is essential in parallel systems to prevent excessive memory contention and support efficient use of shared resources.
2) Prefetch Timeliness L_t: Assesses the promptness of prefetching relative to on-demand memory accesses, ensuring that data is available precisely when needed. Timeliness is vital for maintaining low latency in parallel architectures where delayed data can create bottlenecks across cores.
3) Cache Pollution P_t: Estimates the interference caused by prefetched data in the L2 cache, as excessive or poorly targeted prefetching can displace valuable data needed by other cores. Controlling pollution is crucial to balance data residency with minimal cache turnover, a key factor in the efficient operation of multi-core systems.

To keep the RL model lightweight and suitable for real-time application, S_t is discretized. We denote this discretized state as $\hat{S}_t = (\hat{A}_t, \hat{L}_t, \hat{P}_t)$. Specifically:

1) Prefetch Accuracy $\hat{A}_t$ is categorized into $\{Low, Mid, High\}$ based on accuracy thresholds A_{high} and A_{low}.
2) Prefetch Timeliness $\hat{L}_t$ is categorized as $\{Late, \text{Not-Late}\}$, using a timeliness threshold T_{lateness}.
3) Cache Pollution $\hat{P}_t$ is classified as $\{Polluting, \text{Not-Polluting}\}$ by comparing against a pollution threshold $T_{\text{pollution}}$.

These discretized metrics allow APAC to make precise, resource-conscious adjustments to prefetch aggressiveness. Since parallel system architectures demand efficient resource utilization across shared memory and caches, this design ensures the controller adapts effectively within the limited hardware storage constraints, balancing the prefetch aggressiveness with core-specific demands to sustain system-wide performance.

2) Action: For a given discrete state $\hat{S}_t$, APAC's actions a_t are encoded as $\langle 0, 1, 2 \rangle$, representing adjustments to prefetcher aggressiveness. Actions 0, 1, and 2 correspond to increasing, decreasing, and maintaining the current prefetch aggressiveness, respectively. This streamlined action space is advantageous in parallel system architectures, where minimizing computational overhead and reducing latency is critical for optimal performance. The limited action set enhances APAC's efficiency by reducing the complexity of the decision-making process, enabling faster real-time adjustments crucial for managing the resource contention and interdependencies characteristic of multi-core systems.

Given constraints in prefetch queue resources and Miss Status Handling Registers (MSHRs) within the L2 cache, excessive aggressiveness is often infeasible. To account for these hardware limitations, two thresholds are implemented to cap the maximum and minimum allowable prefetch degrees. The range of prefetch degrees is set between 3 and 8, ensuring that adjustments respect hardware constraints. This range prevents overwhelming shared resources, promoting stability in parallel system architectures while maintaining efficient prefetching.

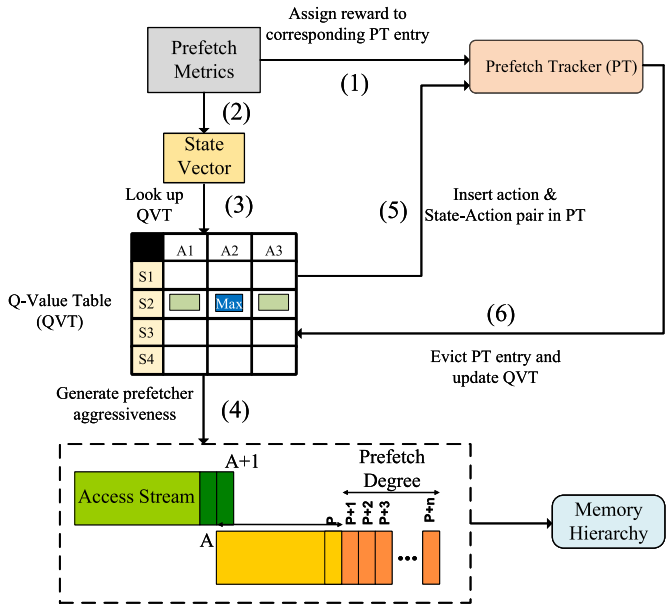


Fig. 4. Overview of APAC.

3) Reward Scheme: The reward scheme defines APAC's objectives by setting incentives to balance performance improvements with bandwidth efficiency, a vital balance in parallel system architectures where bandwidth is often a bottleneck. APAC's end-to-end prefetch control is designed to dynamically tune aggressiveness in response to the observed performance and bandwidth trade-offs. Six distinct reward levels guide the controller's policy:

1) *Performance Increased (R_{IN}):* This reward is given when an action leads to improved program performance. It is further divided into two sub-levels: performance increased with low bandwidth usage (R_{IN}^{L}) and performance increased with high bandwidth usage (R_{IN}^{H}).
2) *Performance Unchanged (R_{FIX}):* Assigned when an action maintains current program performance. It is categorized into: performance unchanged with low bandwidth usage (R_{FIX}^{L}) and performance unchanged with high bandwidth usage (R_{FIX}^{H}).
3) *Performance Degraded (R_{DE}):* Assigned when an action results in degraded program performance, with sub-levels: performance degraded with low bandwidth usage (R_{DE}^{L}) and performance degraded with high bandwidth usage (R_{DE}^{H}).

By increasing or decreasing reward values, APAC strengthens or discourages specific adjustments, guiding it to optimize the balance between performance and bandwidth efficiency. The sub-level rewards are key to defining APAC's aggressiveness control policy based on feedback regarding memory bandwidth usage, an important consideration in parallel architectures where memory access patterns vary widely across cores.

To quantify the performance-bandwidth trade-off and determine optimal aggressiveness adjustments, delta IPC and memory bandwidth usage are evaluated at each timestep. Delta IPC, a measure of performance change, is classified based on thresholds τ_{hi} and τ_{lo}. We classify the delta IPC as unchanged, if it is in τ_{hi} and τ_{lo}. If the delta IPC is greater than τ_{hi}, it is classified as increased, indicating a performance improvement. Conversely, if the delta IPC is lower than τ_{lo}, it is classified as decreased. The values of τ_{hi} and τ_{lo} are dynamically adjusted according to the workload and system performance, with typical ranges observed between 0 and 0.1 for τ_{hi}, and between -0.01 and 0.01 for τ_{lo}. This adaptive approach ensures that the thresholds reflect real-time system behavior and optimize prefetch aggressiveness accordingly.

To maximize APAC's performance benefits within parallel system architectures, dynamic thresholds are employed to evaluate delta IPC based on current workload characteristics. This dynamic threshold adjustment enhances APAC's responsiveness to the diverse and fluctuating demands of parallel systems, described in detail in Section VI-D.

B. Apac Framework

Fig. 4 provides a high-level overview of APAC. APAC is primarily composed of two hardware structures: the Q-value Table (QVT) and the Prefetch Tracker (PT). The QVT records Q-values for all observed state-action pairs, enabling APAC to efficiently select optimal actions. The PT, on the other hand, stores the most recent state and action taken by APAC, maintaining a history of system interactions to refine future decision-making.

In parallel system architectures, managing prefetch aggressiveness becomes more complex due to the increased resource contention and inter-thread communication overhead. APAC is designed to address these challenges by dynamically adjusting the prefetcher aggressiveness based on the interaction between cores. Specifically, at each update interval, defined by a fixed number of instructions, APAC updates its prefetch aggressiveness decisions. The duration of the update interval determines the frequency of these adjustments, balancing the need for timely responsiveness with the computational cost of frequent updates. At each update interval, APAC allocates a reward to the PT entry based on memory bandwidth usage and delta IPC (1), both of which are critical in a parallel setting as they reflect the load on shared resources and the impact on overall system throughput. APAC then updates its state vector, which includes metrics such as prefetch accuracy, timeliness, and cache pollution (2). These factors are particularly influential in parallel systems where inefficient prefetches can lead to increased cache contention and reduced performance across multiple cores. By referencing the QVT, APAC identifies the action with the highest Q-value for the given state vector (3), selecting the most beneficial prefetch aggressiveness adjustment for the current system state. Here, APAC's ability to choose optimal actions based on cumulative learning is essential to balance resource utilization across cores.

The selected action adjusts the prefetcher aggressiveness, determining the frequency and scale of prefetch requests to the memory hierarchy (4). This adjustment helps mitigate resource contention, allowing APAC to support efficient prefetching even in high-load parallel configurations. The action and state vector are then recorded in the PT (5), which enables APAC to update Q-values based on subsequent outcomes. When a PT entry is

Algorithm 1: APAC's Reinforcement Learning-Based Prefetcher Aggressiveness Controlling Algorithm.

procedure Initialize
  initialize QVT: Q(S, A) $\leftarrow \frac{1}{1-\gamma}$
  clear PT
procedure Train _ and_Predict(Aggre)
  entry $\leftarrow$ search$_{PT}$(Acc, Late, Pollu) /* Search PT with prefetch accuracy, timeliness, and cache pollution */
  S $\leftarrow$ get _ state() /* Get the current state */
if rand() $\leq \epsilon$ **then**
  action $\leftarrow$ select _ random_action() /* With a low probability ϵ, select a random action */
else
  action $\leftarrow$ $argmax_a$Q(S, a) /* Otherwise, select the action with the highest Q value */
end if
  pref _ degree = pref _ degree + Aggre[action] /* Adjust prefetch aggressiveness based on the selected action */
  $S_1 \leftarrow$ S /* Set the current state to S1 */
  Perform _ Action() /* Perform the prefetch action */
  $R \leftarrow$ Reward(delta _ ipc, usage _ bw) /* Compute reward based on delta IPC and memory bandwidth usage */
  $S_2 \leftarrow$ get _ state() /* Get the next state S2 */
  Q(S_1, action) $\leftarrow$ Q(S_1, action) + α [$R + \gamma\ argmax_a$ Q(S_2, a) - Q(S_1, action)] /* SARSA Q-value update */
  entry $\leftarrow$ create _ PT_entry(S_1, action) /* Create a new PT entry with the current state and action */
  insert _ PT(entry) /* Insert the new PT entry */

evicted, APAC uses the state-action pair and the reward from the evicted entry to update the corresponding Q-value in the QVT (6), continuously refining its strategy.

By integrating these mechanisms, APAC effectively enhances performance in parallel system architectures, where dynamically balancing prefetch aggressiveness is key to optimizing memory bandwidth usage and maintaining high inter-core data locality.

C. RL-Based Prefetcher Aggressiveness Control Algorithm

Algorithm 1 outlines APAC's prefetcher aggressiveness control algorithm, optimized for parallel system architectures where multiple cores share and compete for memory resources. Initially, each entry in the Q-value Table (QVT) is set to the maximum Q-value $\frac{1}{1-\gamma}$ to encourage exploration, and the Prefetch Tracker (PT) is cleared. At each update interval, APAC calculates memory bandwidth usage and delta IPC, both key metrics for reflecting resource load in parallel systems. Based on these values, a reward is assigned to guide action selection toward efficient resource use. APAC then extracts the current state vector, which includes prefetch accuracy, timeliness, and cache pollution—factors critical in multi-core settings to avoid cache contention and maximize data locality. APAC identifies the action with the highest Q-value for the state vector and adjusts prefetcher aggressiveness accordingly, balancing prefetch frequency with shared memory demands. A new PT entry containing the current state and selected action is subsequently inserted, evicting an existing entry if necessary to maintain capacity constraints. Using the SARSA update method described in Section III-A1, APAC updates the Q-value of the evicted state-action pair, refining its decisions over time.

This RL approach enables APAC to adaptively manage prefetch aggressiveness, optimizing performance in parallel system architectures by balancing system throughput and resource contention.

TABLE I
BASIC APAC CONFIGURATION

Reward Level Values	$R^L_{IN} = 20$, $R^H_{IN} = 12$, $R^L_{FIX} = 12$, $R^H_{FIX} = 6$, $R^L_{DE} = -8$, $R^H_{DE} = -14$
Hyperparameters	$\alpha = 0.0065$, $\gamma = 0.55$, $\varepsilon = 0.3$
Thresholds	$A_{high} = 0.75$, $A_{low} = 0.40$, $T_{lateness} = 0.01$, $T_{pollution} = 0.005$

D. Detailed Design of APAC

In this subsection, we describe how APAC collects state information (Section IV-D1) and tracks bandwidth utilization (Section IV-D2) in the context of parallel system architectures, where efficient memory management and minimal contention are crucial for maintaining performance across multiple cores. To determine the parameters in Table I, we used an empirical tuning process along with a performance-driven approach. This method systematically explores the design space of APAC to balance its adaptability and stability across different workloads, resulting in the selection of optimal values for the parameters. Table I shows the basic configuration.

1) State Information Collection: In parallel system architectures, precise state information is essential for APAC to adjust prefetch aggressiveness effectively. The following components provide detailed metrics to guide these adjustments.

Prefetch Accuracy: To evaluate the effectiveness of prefetches across cores, APAC includes a pref-bit in the tag-store entry of each prefetched block. Prefetch accuracy is measured using two hardware counters: pref-total counts all issued prefetches, while used-total counts only those that were subsequently requested by a core. The accuracy is calculated as the percentage of used-total to pref-total, which helps reduce unnecessary prefetches that can otherwise lead to cache pollution and contention in shared resources.

Prefetch Lateness: In parallel systems, delayed prefetches can stall critical data flow among cores. Each entry in the L2 cache Miss Status Holding Register (MSHR) contains a pref-bit indicating a prefetch request. A prefetch is considered "late" if the prefetched address is demanded by a core before the prefetch is filled. The late-total counter increments for each late prefetch, calculated as the percentage of late-total to used-total. This metric allows APAC to adjust prefetching aggressiveness to minimize delays, which is especially important for maintaining data availability across cores.

Prefetcher-Induced Cache Pollution: In parallel architectures, excessive prefetching can increase cache contention, leading to cache pollution that affects all cores. To measure cache pollution, APAC tracks prefetch-induced demand misses using a Bloom filter, following the approach described in FDP [11]. When a block is brought into the L2 cache due to a demand miss and subsequently evicted by a prefetch, its address is hashed and recorded in the Bloom filter. Upon a subsequent demand miss, the block's address is checked against the Bloom filter. If a match is found, it indicates that the miss was caused by a prefetch eviction. APAC employs two counters for each core: the pollution-total counter, which tracks the number of demand misses resulting from prefetch-induced evictions, and the demand-total counter, which tracks all demand misses. Cache pollution for each core is calculated as the ratio of pollution-total to demand-total, providing an estimate of the percentage of demand misses caused by prefetching. Although APAC is based on the mechanism described in FDP, it refines this approach to handle multi-core environments by calculating cache pollution independently for each core. This refinement ensures that the pollution measurement accurately reflects the impact of prefetching from each core.

These state information metrics are vital for APAC's adaptability in parallel system architectures, where efficient resource sharing is critical. By tracking prefetch accuracy, lateness, and cache pollution across cores, APAC can dynamically adjust its prefetching strategy to optimize performance and mitigate contention within shared resources like cache and memory bandwidth. This approach is especially effective in parallel environments, as it enables APAC to address the unique challenges of inter-core dependencies and shared resource contention, ensuring that prefetching operations contribute positively to overall system throughput and scalability.

2) Tracking Bandwidth Utilization: Efficient bandwidth management is vital in parallel systems to prevent one core's memory access from negatively impacting others. APAC tracks memory bandwidth utilization using a counter in the memory controller, which measures DRAM Column Access (CAS) commands over a window of $4 \times$ tRC cycles (where tRC is the minimum interval between two row activations). To account for changes in workload demand over time, the counter is halved at the end of each window, introducing hysteresis into the tracking mechanism.

In parallel architectures, where shared bandwidth is a limited resource, the peak DRAM bandwidth and maximum CAS command count per tRC cycle are influenced by the number and width of memory channels [47]. APAC divides this bandwidth utilization into quartiles (25%, 50%, and 75% of peak capacity), comparing the counter's value to these thresholds in each tRC cycle. If the counter exceeds the 75% threshold, indicating high bandwidth usage, APAC classifies it as high bandwidth utilization, which could indicate potential contention across cores. Conversely, if usage is below the 75% threshold, it is classified as low.

These classifications allow APAC to adjust its prefetching policy based on real-time bandwidth conditions, ensuring that prefetch aggressiveness is optimized to support parallel cores without overwhelming shared memory resources. This dynamic approach helps maintain system throughput by balancing prefetch requests with available bandwidth, a critical factor in parallel system architectures.

E. Coordinated Cache Partitioning and Prefetcher Aggressiveness Control

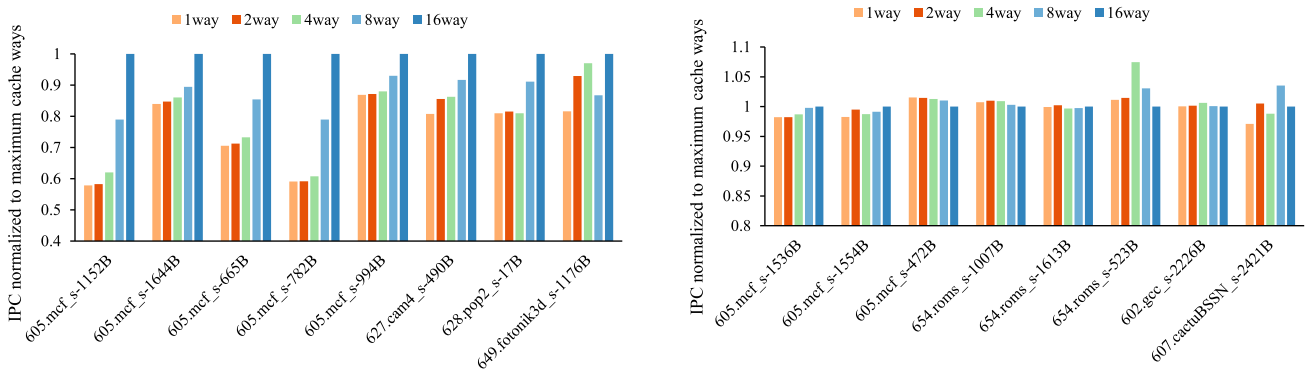
1) Interaction Between Cache Partitioning and Prefetch Control: Cache partitioning is a widely adopted technique to mitigate inter-core interference in parallel system architectures, where multiple cores share a common cache. Utility-based Cache Partitioning (UCP) [48] is a classical method that partitions shared cache by allocating more cache ways to applications with higher cache utility, optimizing cache usage for applications that benefit most from additional cache space.

In parallel system architectures, both cache partitioning and prefetcher aggressiveness control are critical to enhancing multi-core processor performance. To examine the interaction between fine-grained aggressiveness control and cache partitioning, we conducted experiments adjusting the number of LLC ways allocated to applications, from 1 to 16, with APAC enabled throughout. These experiments allowed us to model how cache partitioning influences APAC's prefetcher aggressiveness control.

Fig. 5 illustrates the slowdown for different SPEC CPU2017 applications. The slowdown is calculated by measuring the IPC when an application uses the full cache (16 ways) with APAC enabled. Results show that LLC allocation affects applications differently. For example, applications in Fig. 5(a) experience performance degradation as their effective LLC size decreases, particularly when APAC is enabled. These applications are sensitive to cache size and may experience cache contention, highlighting the importance of balancing cache space and prefetcher activity across cores.

Conversely, applications in Fig. 5(b) exhibit minimal performance degradation even as LLC size decreases. These applications benefit from aggressive prefetching and can maintain near-optimal performance with fewer cache ways. By adjusting prefetcher aggressiveness, APAC offsets potential performance loss from limited cache allocation, allowing the remaining cache to be assigned to cache-sensitive applications. However, inadequate cache allocation can cause prefetching to negatively impact performance in certain applications, as aggressive prefetching may increase contention and reduce effective cache usage for neighboring cores.

In our experiments, few programs show performance improvements when cache capacity is reduced. The APAC



(a) Sensitive applications (b) Non sensitive applications

Fig. 5. Application slowdown when varying the number of cache-ways allocated, if APAC is enabled.

framework dynamically adjusts prefetch aggressiveness to prevent cache overuse and cache pollution. By reducing prefetch degree, APAC ensures more efficient utilization of the available cache, allowing some applications to maintain high performance even with reduced cache allocation. This demonstrates that, with proper prefetch aggressiveness adjustment, performance can still be optimized even with limited cache resources. In parallel system architectures, coordinating cache partitioning with prefetcher aggressiveness control is crucial. A cache partitioning-aware prefetcher aggressiveness controller is needed to optimize the balance of shared resources, ensuring that cache-sensitive applications receive sufficient cache allocation while prefetch-friendly applications maintain performance with fewer cache ways. This approach minimizes inter-core interference, enabling efficient resource sharing and improved performance across all cores in a parallel system.

2) Partitioning Aware Prefetcher Aggressiveness Control: To enhance resource management in parallel system architectures, we propose a coordinated approach to cache partitioning and prefetcher aggressiveness control. This coordination is critical in multi-core systems, where cache resources are shared and must be dynamically balanced to optimize performance across cores. We extend APAC to create APAC _ CP, a cache partitioning-aware prefetcher aggressiveness controller that incorporates cache allocation data directly into its state function. By doing so, APAC _ CP learns the combined impact of prefetcher aggressiveness and cache allocation adjustments on IPC, enhancing its learning effectiveness.

To communicate the aggressiveness controller about cache allocation changes, we discretize the changes information as $\hat{C}_t \in C = \{Allocation(1), Not-Change(0), Deallocation(-1)\}$, so that the state function turns to be $S_t = (A_t, L_t, P_t, C_t) \in R^4$ as shown in Fig. 6. $Allocation$, $Not-Change$ and $Deallocation$ represent LLC allocation to a core is increased, unchanged and decreased, respectively. The reward scheme and action space of APAC _ CP are the same as those of APAC, as described in Algorithm 2. This approach enables APAC _ CP to manage both cache partitioning and prefetching in a way that dynamically aligns with the needs of parallel system architectures, improving multi-core performance by reducing inter-core interference and optimizing resource utilization across all cores.

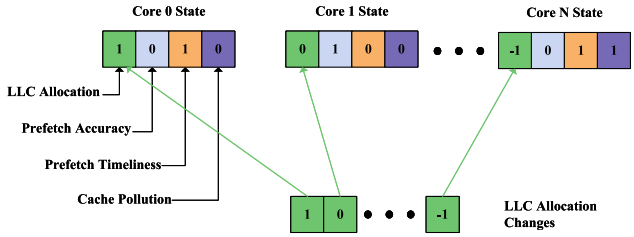


Fig. 6. State function of APAC _ CP.

F. Implementation Considerations for APAC in Parallel System Architectures

In designing APAC for parallel system architectures, we emphasize a lightweight and efficient implementation that aligns with the demands of multi-core environments. To achieve this, the RL state space is discretized, minimizing the complexity of the Q-value Table (QVT) for fast, low-latency predictions. Compared to Pythia, APAC features a more compact state and action space, reducing both memory and computational overhead. As shown in Table II, APAC's storage footprint is limited to 1.58 KB, making it suitable for resource-constrained architectures.

The streamlined design of APAC not only reduces area but also lowers power consumption—a critical consideration in parallel system architectures, where multiple cores must efficiently share limited power and thermal budgets. We assess APAC's dynamic power in Section VI-G and find that it offers notable performance gains with minimal power requirements. This efficiency enables APAC to promptly adjust prefetcher aggressiveness without burdening the processor, making it highly suitable for real-world multi-core processors where scalability and resource sharing are essential.

In parallel system architectures, APAC's ability to adjust prefetcher aggressiveness dynamically allows for responsive resource management across cores, reducing inter-core interference and optimizing cache and memory bandwidth utilization. Consequently, APAC's low overhead and rapid adaptability

Algorithm 2: APAC_CP's Cache Partitioning Aware Prefetcher Aggressiveness Controlling Algorithm.

procedure Initialize
  initialize QVT: Q(S, A) $\leftarrow \frac{1}{1-\gamma}$
  clear PT
procedure Train _ and_Predict(Aggre, Cache _ Alloc)
  entry $\leftarrow$ search $_{PT}$(Acc, Late, Pollu, Cache_ Alloc) /* Search PT with prefetch accuracy, timeliness, cache pollution, and cache allocation */
  S $\leftarrow$ get _ state() /* Get the current state, now including cache allocation information */
if rand() $\leq \epsilon$ **then**
  action $\leftarrow$ select _ random_action() /* With a low probability ϵ, select a random action */
else
  action $\leftarrow argmax_a$Q(S, a) /* Otherwise, select the action with the highest Q value */
end if
  pref _ degree = pref _ degree + Aggre[action] /* Adjust prefetch aggressiveness based on the selected action */
  $S_1 \leftarrow$ S /* Set the current state to S1 */
  Perform _ Action() /* Perform the prefetch action */
  $R \leftarrow$ Reward(delta _ ipc, usage _ bw) /* Compute reward based on delta IPC and memory bandwidth usage */
  $S_2 \leftarrow$ get _ state() /* Get the next state S2 */
  Q(S_1, action) $\leftarrow$ Q(S_1, action) + α [R + γ $argmax_a$ Q(S_2, a) - Q(S_1, action)] /* SARSA Q-value update */
  entry $\leftarrow$ create _ PT_entry(S_1, action) /* Create a new PT entry with the current state and action */
  insert _ PT(entry) /* Insert the new PT entry */

TABLE II
ESTIMATION OF APAC STORAGE OVERHEAD

Structure	Description	Size
QVT	# entries = 36, # Entry size = Q-value width (16b)	72 B
PT	# size = state (4b) + action index (2b) + reward (5b)	1.37 B
tag-store pref-bit	# 8192 blocks * 1 bit/block	1 KB
Bloom filter	# 4096 entries * 1 bit/block	0.5 KB
MSHR pref-bit	# 32 entries * 1 bit/entry	4 B
counters	# 6 counters * 16 bits/counter	12 B

TABLE III
BASELINE CONFIGURATION

Core Parameters	256 entry ROB, 4-wide, 128/72-entry LQ/SQ
Branch Pred.	Bimodal, 20 cycle mispredict penalty
L1/L2 Caches	Private, 48KB/512KB, 64B line, LRU, 16/32 MSHRs, 4 cycles/14 cycles
LLC	2MB/core, 64B line, 16-way, LRU, 64 MSHRs per LLC Bank, 34 cycles
Main Memory	1C: Single channel, 1 rank/channel; 4C: Dual channel, 2ranks/channel; 8C: Quad channel, 2 ranks/channel; 8 banks/rank, 1600 MT/s

make it well-suited for integration into actual processors, enhancing performance in multi-core environments.

V. PERFORMANCE ANALYSIS

A. Experimental Settings

We evaluate APAC employing the simulation framework provided by the second JILP Cache Replacement Championship (CRC2), which is built on ChampSim [49]. Table III displays the parameters of our simulated system.

Hardware Prefetcher: We use IP-based stride prefetcher (IP _ stride prefetcher) as the baseline hardware data prefetcher, which is commonly used in current commercial processors [50] [51]. The IP-based stride prefetcher improves performance by analyzing access patterns within a spatial region. It uses IP information to generate prefetch requests based on detected patterns.

Prefetcher Aggressiveness Controller: We compare APAC with Feedback Directed Prefetching (FDP) [11] and the IP _ stride prefetcher using the built-in method of prefetcher aggressiveness controlling in IPCP [23] (IP _ stride-IPCP). FDP dynamically selects between five different aggressiveness levels based on rules, from very conservative to very aggressive. IPCP measures the prefetch accuracy once in a fixed number of prefetch fills, and compares the accuracy with two thresholds to adjust the prefetch aggressiveness.

Benchmarks: We assess APAC using the SPEC CPU2017 [52] benchmarks, relying on sim-point traces offered by DPC-3. For conciseness, we specifically elaborate on the memory-intensive benchmarks (46 traces with LLC MPKI$\geq$ 1). For single-core configuration, we warmed-up caches for 50M (50 million) instructions and assessed performance for the subsequent 200 million instructions.

Multi-core workloads: For multi-core simulations, we simulate 30 homogeneous mixes and 60 heterogeneous mixes. In the case of homogeneous mixes, each mix consists of four identical memory-intensive traces, where all traces are executed concurrently across the cores. Each trace runs until at least 250 M instructions are executed, and if a trace finishes early, it is reiterated until all traces complete 250 M instructions. For heterogeneous mixes, the traces vary across cores, and the same process is followed to ensure concurrent execution of the traces.

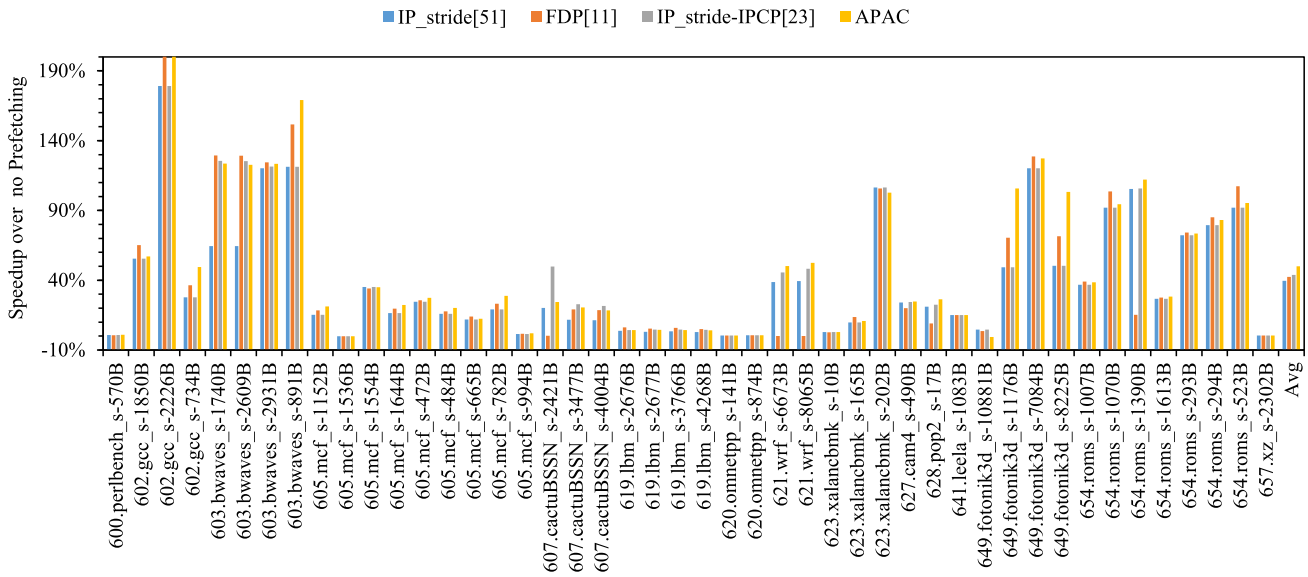


Fig. 7. Performance improvement in single-core workloads.

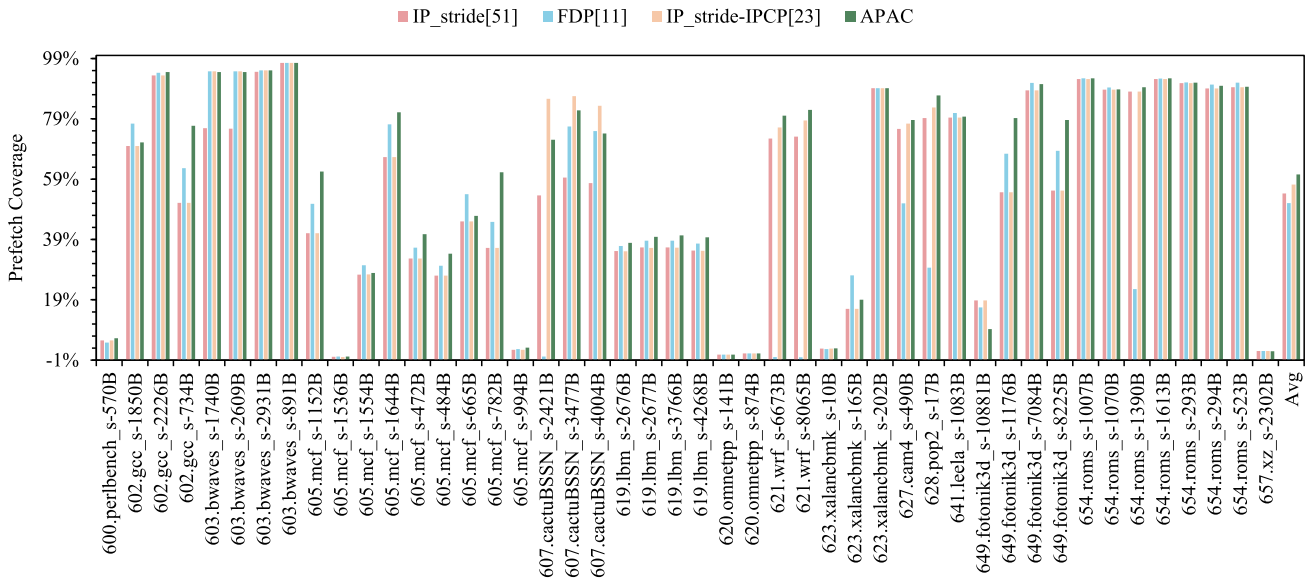


Fig. 8. Demand misses covered by prefetcher at LLC.

B. Experiment Results

1) Single-Core Performance: We perform our evaluation for APAC to adjust the aggressiveness of the IP _ stride prefetcher, comparing with IP _ stride prefetcher (IP _ stride), Feedback-Directed Prefetching (FDP) and the IP _ stride prefetcher using the built-in method of prefetcher aggressiveness controlling in IPCP (IP _ stride-IPCP). Fig. 7 shows the experimental results. From Fig. 7, it can be seen that APAC has the best performance across all configurations on average. Compared with the baseline without prefetching, APAC improves the performance by 49.92% on average. APAC provides 10.43%, 7.44% and 6.14% performance improvement on average over the IP _ stride prefetcher, FDP, and IP _ stride-IPCP respectively. The adjustment policies of FDP and IP _ stride-IPCP are adjusted based on manual rules, while APAC employs RL to find the optimal aggressiveness controlling policy, aiming for the most significant long-term performance advantage. However, for certain benchmarks like 623.xalancbmk _ s-10B and 649.fotonik3d _ s-10881B, the improvements are minor due to their memory access patterns being less sensitive to prefetching. In these cases, the existing prefetching mechanisms are already well-tuned to the workload's access patterns, leaving little room for further optimization.

Fig. 8 illustrates the single-core prefetch coverage for each configuration in the LLC. In comparison to rule-based prefetcher aggressiveness controllers, APAC enhances prefetch coverage. On average, APAC delivers 6.34%, 9.54%, and 3.37% more coverages than IP _ stride, FDP, and IP _ stride-IPCP, respectively. In addition to improving coverage, APAC also enhances

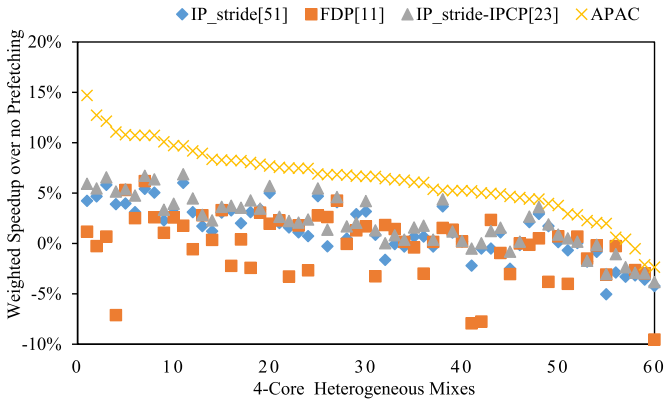


Fig. 9. Weighted speedup for 4 cores under heterogeneous workloads.

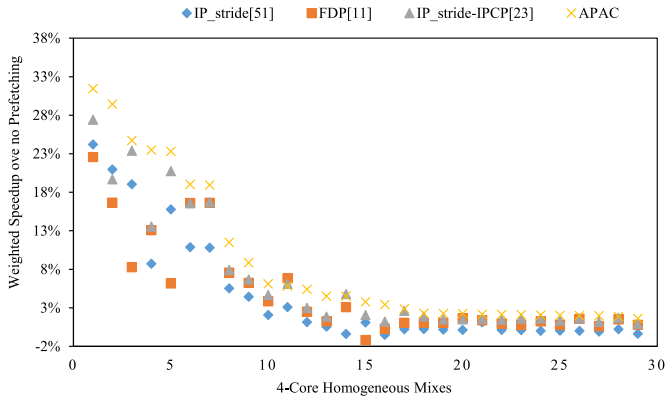


Fig. 10. Weighted speedup for 4 cores under homogeneous workloads.

prefetch accuracy and timeliness. It outperforms IP _ stride, FDP, and IP _ stride-IPCP by 13.30%, 7.39%, and 5.46% in accuracy. In terms of timeliness, APAC improves by 10.28%, 5.80%, and 3.83% over IP _ stride, FDP, and IP _ stride-IPCP, respectively. Furthermore, the distribution of prefetch degrees reveals APAC's ability to dynamically adjust aggressiveness, with average distributions of 5.01%, 9.15%, 10.53%, 34.76%, 21.02%, and 19.53% for degrees 3 through 8. These results demonstrate APAC's effectiveness in balancing performance improvements with efficient resource utilization.

2) Multi-Core Performance: In parallel system architectures, where multiple cores compete for shared resources such as memory bandwidth and cache space, balancing prefetch aggressiveness is essential for maintaining overall system efficiency. Fig. 9 illustrates that APAC outperforms IP _ stride, FDP, and IP _ stride-IPCP by 5.10%, 6.73%, and 4.22% on average in the 4-core system, respectively. We evaluated the proportion of runtime where bandwidth usage exceeded 75% of the peak bandwidth. APAC's proportion in this high-bandwidth usage category was 78.60%, which is a 2.96% reduction compared to IP _ stride. This reduction in memory bandwidth usage not only indicates more efficient utilization of available bandwidth but also contributed to a 5.10% performance improvement over IP _ stride. These results demonstrate APAC's ability to optimize memory bandwidth while simultaneously enhancing system performance in multi-core environments, highlighting its effectiveness in balancing resource contention and overall throughput.

To further evaluate APAC's performance, we simulate an 8-core system equipped with four DDR4-2400 DRAM channels. In this setup, APAC outperforms the IP _ stride prefetcher, FDP, and IP _ stride-IPCP by 2.91%, 8.92%, and 2.24%, respectively.

Additionally, to assess APAC's performance in homogeneous multi-core environments, we conducted experiments using 30 homogeneous mixes derived from SPEC CPU2017 traces. The results for homogeneous workloads further demonstrate APAC's effectiveness. Specifically, APAC outperforms the IP _ stride prefetcher, FDP, and IP _ stride-IPCP by 4.18%, 3.66%, and 1.88%, respectively. These results highlight APAC's ability to dynamically optimize prefetcher aggressiveness, even in homogeneous multi-core environments, and improve overall system performance. As shown in Fig. 10, APAC outperforms the other prefetching schemes in this scenario, emphasizing its robustness and adaptability across different workload configurations.

In parallel system architectures, the lack of coordinated prefetcher adjustments, as seen with FDP, frequently leads to significant resource contention. Without synchronized control, multiple prefetchers may demand high bandwidth simultaneously, creating bottlenecks and reducing overall system performance. Furthermore, IP _ stride-IPCP fails to adequately account for the impact of its aggressiveness on shared cache and memory bandwidth within multi-core environments, resulting in performance degradation as the number of cores increases.

In contrast, APAC employs RL to dynamically adjust each prefetcher's aggressiveness, guided by real-time feedback on bandwidth usage and performance. This adaptive, data-driven approach is optimized for parallel system architectures, where varying workload demands and inter-core dependencies complicate resource management. By continuously balancing prefetch aggressiveness with bandwidth consumption, APAC maximizes shared resource efficiency and system performance, presenting a robust solution tailored to the challenges of prefetching in complex, multi-core environments.

3) Benefit of Cache Partitioning Awareness: Effective resource management is crucial for optimizing performance in parallel system architectures, particularly in multi-core systems where multiple cores share resources such as cache and memory bandwidth. To further assess the impact of cache partitioning-aware prefetcher aggressiveness control on performance, we conducted experiments using several configurations. In the IP _ stride + UCP configuration, the UCP cache partitioning scheme is enabled, but the APAC prefetching algorithm is turned off, with only the IP _ stride prefetcher operating, without any partitioning-aware mechanisms. In contrast, the APAC _ CP + UCP configuration integrates our proposed Partitioning-aware Prefetcher Aggressiveness Control (APAC _ CP), which dynamically adjusts prefetcher aggressiveness based on the current cache partitioning scheme, while also utilizing UCP for cache partitioning. The APAC + UCP setup enables UCP but relies on the original APAC prefetching algorithm, which lacks the partitioning-aware enhancements.

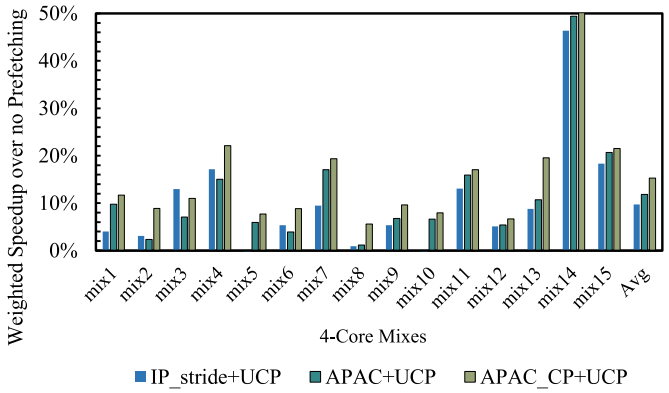


Fig. 11. Weighted speedup comparison for 4 cores with APAC and APAC _ CP using cache partitioning.

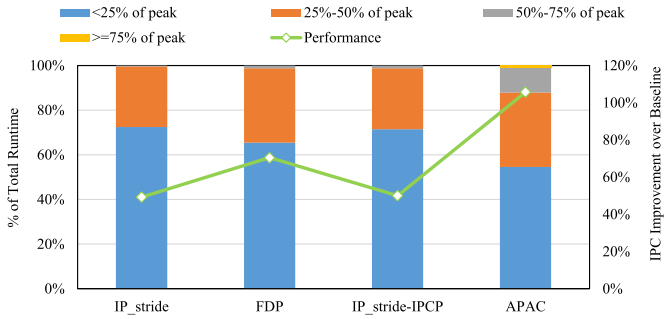


Fig. 12. Performance and main memory bandwidth usage in 649.fotonik3d _ s-1176B.

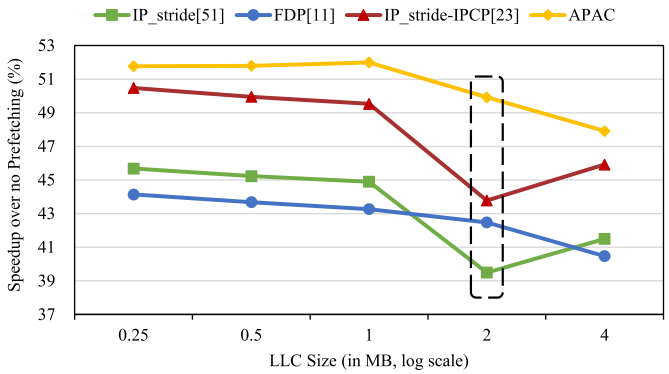


Fig. 13. Average performance improvements for 46 SPEC17 traces in systems with varying LLC size.

The experimental results show that APAC _ CP + UCP outperforms APAC + UCP by an average of 3.45% in multi-core performance. Additionally, APAC _ CP + UCP achieves 5.60% better performance compared to IP _ stride + UCP, as illustrated in Fig. 11. This dynamic adaptability is crucial in parallel systems, where cache and memory bandwidth allocation can vary based on workload distribution and core activity. By integrating cache partitioning awareness with prefetcher aggressiveness control, APAC _ CP ensures more efficient resource utilization, leading to improved multi-core performance and system efficiency.

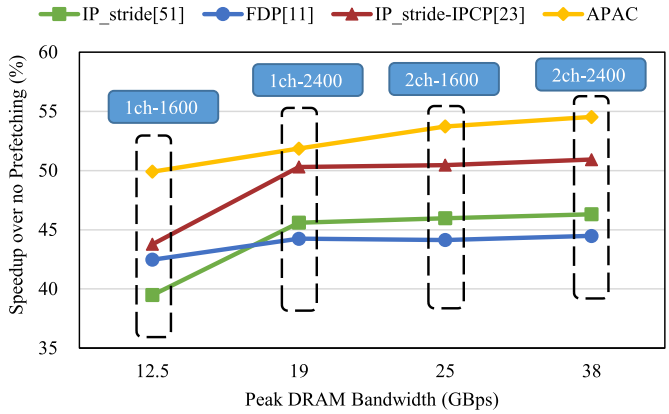


Fig. 14. Average performance scaling with DRAM bandwidth.

VI. Discussion

In this section, we analyze APAC's performance within parallel system architectures, emphasizing its impact on performance and bandwidth efficiency. In these architectures, where cores share key resources such as the LLC and memory bandwidth, optimizing prefetch aggressiveness is critical due to workload variability and resource contention. We evaluate APAC's adaptability across varying LLC sizes, memory bandwidth, and power constraints, focusing on its RL parameters, including the dynamic reward structure and timestep settings. This analysis underscores APAC's role in optimizing prefetching strategies for enhanced resource management in parallel environments.

A. Performance and Bandwidth Consumption

Fig. 12 shows the proportion of the total runtime workloads takes in disparate bandwidth usage buckets in the primary y-axis, and the IPC speedup of IP _ stride prefetcher, FDP, IP _ stride-IPCP, and APAC in the secondary y-axis in one example trace, 649.fotonik3d _ s-1176B. It can be observed that 649.fotonik3d _ s-1176B consumes less than half of the peak DRAM bandwidth for a high proportion of runtime. In this case, APAC has only a proper memory bandwidth consumption and provides 105.73% performance improvements over the baseline. In this parallel system architecture, where multiple cores share the limited memory bandwidth and other critical resources, APAC's ability to dynamically adjust prefetch aggressiveness based on bandwidth availability becomes crucial.

B. Sensitivity to the LLC Size

Fig. 13 shows the performance improvement of IP _ stride prefetcher, FDP, IP _ stride-IPCP, and APAC averaged over all traces, when the LLC size is changed from 256KB to 4MB. It can be observed that APAC consistently outperforms IP_ stride prefetcher, FDP, and IP _ stride-IPCP across different LLC size configurations. For 256KB (and 4MB) LLC, APAC improves the performance over IP_ stride, FDP, and IP _ stride-IPCP by 6.09% (7.63%), 6.42% (7.45%), and 1.29% (2.00%), respectively. These RL-driven adaptive strategies enable APAC

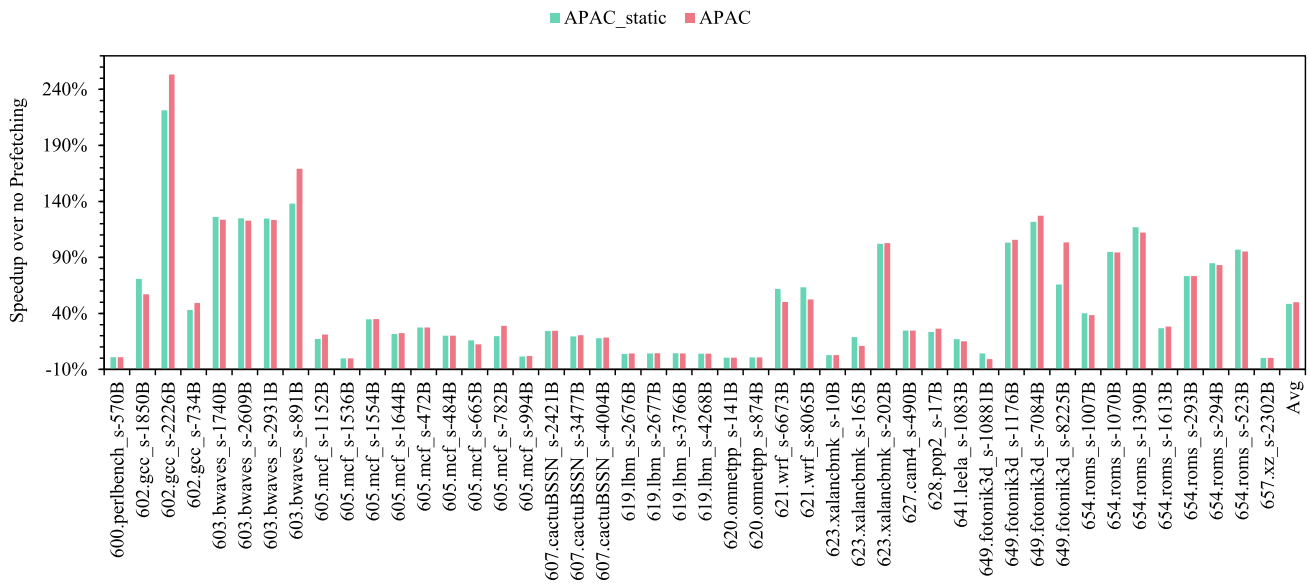


Fig. 15. Performance of the basic and static threshold APAC configurations on the SPEC17 traces.

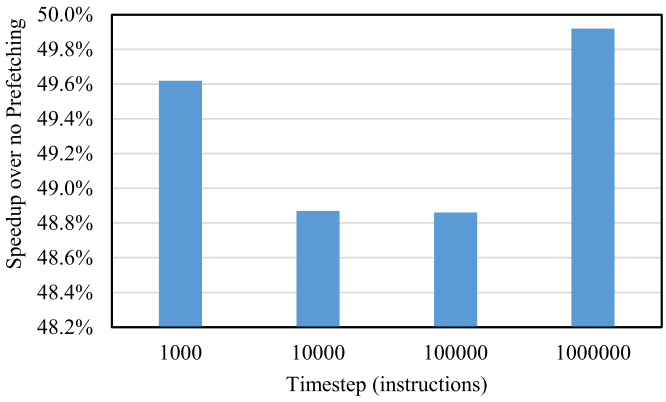


Fig. 16. Performance improvements of APAC under different timestep settings.

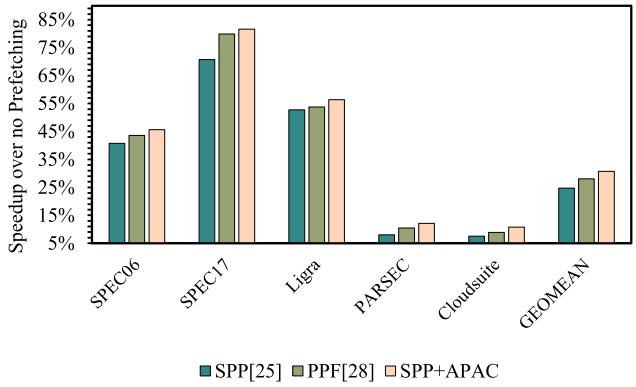


Fig. 17. Performance comparison of SPP, PPF, and APAC on single-core workloads.

to deliver consistent performance gains across various LLC configurations, underscoring its scalability and adaptability in parallel system architectures.

C. Performance Scaling With Memory Bandwidth

We assess APAC's effectiveness across various DRAM bandwidth configurations by conducting experiments that involve scaling the DRAM bandwidth. Fig. 14 shows how the performance improvements of each configuration scale, when we change the DRAM bandwidth from the single-channel DDR4-1600 (with 12.5 GBps bandwidth) to dual-channel DDR4-2400 (with 38 GBps bandwidth) [53]. The performance of APAC scales well with memory bandwidth increase. It outperforms IP_stride from 6.26% to 8.21% when the memory bandwidth is doubled, transitioning from the single-channel DDR4-2400 to the dual-channel DDR4-2400. APAC's RL model adapts prefetch aggressiveness in response to available memory bandwidth, a crucial feature in parallel system architectures where multiple cores may contend for memory resources.

D. Evaluation on Dynamic Reward Structure in APAC

Fig. 15 shows the performance improvements of the basic APAC configuration and APAC with static thresholds on the SPEC CPU2017 traces. We set τ_{hi} and τ_{lo} as two static thresholds in APAC_*static*. Specially, by dynamically adjusting the thresholds for judging delta IPC, APAC achieves up to 37.61% (1.45% on average) performance improvement on top of APAC with a static threshold. We use domain knowledge to tune experimental thresholds dynamically, which can further improve the effectiveness of APAC.

E. Evaluation on Timestep Setting in APAC

To assess the performance of APAC under different timestep settings, we conducted comprehensive tests with timesteps is set to 1000, 10000, 100000, and 1000000 instructions on all traces. As illustrated in Fig. 16, APAC exhibited the average performance improvements of 49.62%, 48.87%, 48.86%, and 49.92%

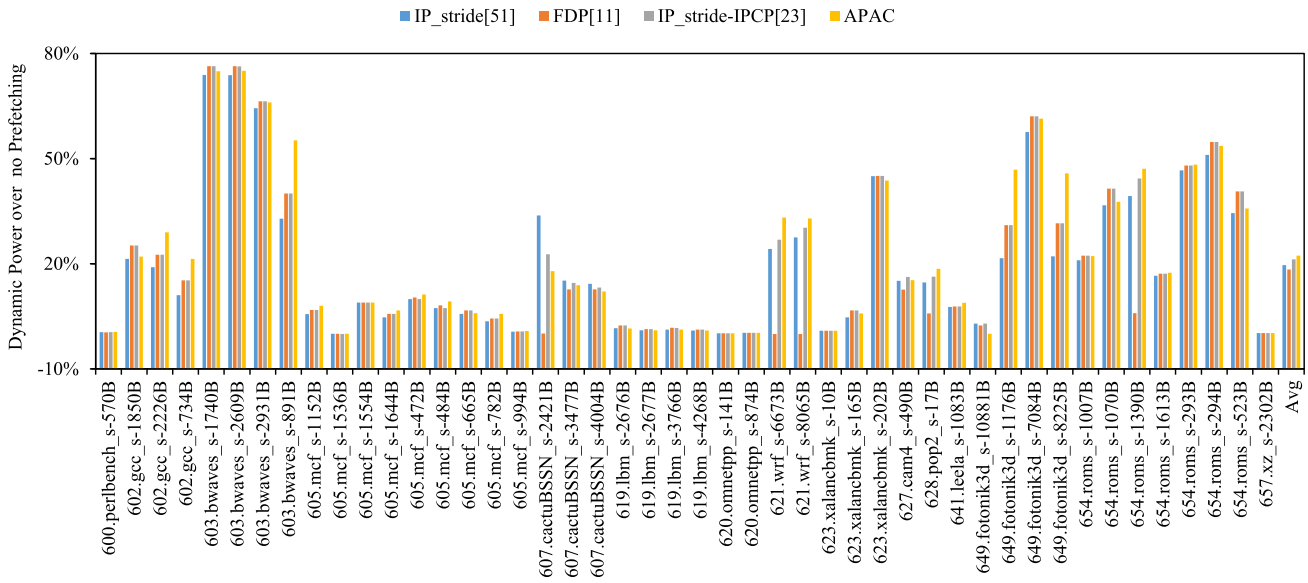


Fig. 18. Processor power consumption of each prefetch policy.

under these settings, respectively. In the design of APAC, striking a balance between performance and computational cost, we select the timestep setting of 1000000 instructions to achieve the optimal performance.

F. Effects of APAC on the SPP Prefetcher

In this section, we evaluate the performance of APAC by comparing it with the SPP, a state-of-the-art L2 prefetcher known for its efficiency in modern processors. Additionally, we compare APAC with PPF, an advanced prefetch control mechanism that optimizes prefetch behavior by filtering unnecessary prefetches based on a perceptron model.

As shown in Fig. 17, our experiments involved a comprehensive set of benchmarks, including SPEC CPU2006 [54], SPEC CPU2017 [52], PARSEC [55], Ligra [56], and Cloudsuite [57], to evaluate the performance of SPP, PPF, and the combined SPP + APAC configuration. These suites include a diverse range of compute-bound, memory-bound, and multithreaded workloads. SPEC CPU2006 and SPEC CPU2017 focus on compute-intensive tasks, PARSEC targets emerging multithreaded applications, Ligra specializes in graph-processing with irregular data structures, and Cloudsuite covers modern cloud workloads like data analytics and web search. Together, these benchmarks provide a comprehensive evaluation of APAC's performance across various application types.

The experimental results indicate that the combination of SPP and APAC delivers an average performance improvement of 30.73% over the baseline system with no prefetching, across a wide range of different workloads. Furthermore, the SPP + APAC configuration outperforms both PPF and SPP by 2.70% and 6.02%, respectively. These findings demonstrate that APAC, a reinforcement learning-driven prefetch aggressiveness controller, effectively adapts to different L2 prefetchers by dynamically adjusting aggressiveness based on real-time system feedback, leading to significant performance improvements.

In conclusion, integrating APAC with the advanced SPP prefetcher results in substantial performance gains, outperforming both SPP and the state-of-the-art prefetch filtering mechanism PPF across all tested workloads. These results highlight APAC's superior capability in adaptively optimizing prefetch behavior, ensuring efficient resource utilization, and providing a flexible, performance-optimized solution for parallel system architectures.

G. Power Overhead

To precisely evaluate the dynamic power consumption of APAC, we use McPAT to compute processor power consumption. We model the configuration of Intel Goldencove to estimate the dynamic power consumed by each policy. Fig. 18 shows the dynamic power consumed by IP _ stride prefetcher, FDP, IP _ stride-IPCP, and APAC, normalized to the no-prefetching baseline. APAC increases processor power consumption by only 2.69% compared to IP _ stride prefetcher on average. However, APAC provides 10.43% higher performance improvement on average over the IP _ stride prefetcher. APAC's ability to dynamically control prefetch aggressiveness based on bandwidth, cache metrics, and workload demands allows it to achieve performance gains with minimal additional power overhead. This balance makes APAC a highly effective and power-conscious solution for parallel system architectures, offering a substantial performance boost with only a modest increase in power, demonstrating its suitability for energy-efficient, high-performance multi-core systems.

VII. Conclusion and the Future Work

In this paper, we propose APAC, an adaptive prefetch aggressiveness controller, which formulates aggressiveness control as a RL process. APAC autonomously learns to adjust prefetcher aggressiveness using multiple metrics of prefetcher

effectiveness and system-level feedback to optimize both performance and bandwidth efficiency. In parallel system architectures, where inter-core dependencies and resource contention are prevalent, APAC's dynamic approach allows it to adapt to diverse workloads and configurations. By managing resource utilization effectively, APAC leverages available DRAM bandwidth to enhance prefetching performance while minimizing adverse effects such as cache pollution and memory contention, which are common in multi-core environments. Through extensive experimental evaluations, APAC has shown superior performance compared to existing prefetcher controllers across a range of workloads and system configurations. In parallel system architectures, where efficient coordination across cores is essential, APAC's scalability allows it to deliver consistent performance improvements by enhancing bandwidth efficiency and overall system throughput. Additionally, APAC's prefetcher aggressiveness controller is versatile, adaptable to various types of hardware prefetchers, making it a solution for enhancing performance in complex, large-scale parallel system architectures.

For the future work in this paper, we will explore additional methods to further refine APAC's adaptability and efficiency in parallel architectures. Specifically, we aim to investigate advanced RL techniques that can incorporate real-time profiling of inter-core interactions, enabling APAC to respond even more accurately to changing workload characteristics. We will also consider incorporating finer-grained cache and memory management strategies to further reduce contention and enhance system-level resource utilization. To further demonstrate the robustness of our approach, we plan to extend our evaluation to include emerging applications, such as Deep Neural Networks (DNNs) and Large Language Models (LLMs), which present unique memory access patterns and computational demands. This will provide insights into the performance and scalability of APAC in addressing the prefetch control challenges posed by these advanced neural network-based workloads.

Acknowledgment

The authors would like to thank the reviewers for their efforts and for providing helpful suggestions that have led to several important improvements in our work. We would also like to thank all teachers and students in our laboratory for helpful discussions.

References

[1] B. Zhang and T. Kosar, "SMURF: Efficient and scalable metadata access for distributed applications," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 12, pp. 3915–3928, Dec. 2022.

[2] M. Bakhshalipour, M. Shakerinava, P. Lotfi-Kamran, and H. Sarbazi-Azad, "Bingo spatial data prefetcher," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, 2019, pp. 399–411.

[3] G. Vavouliotis, G. Chacon, L. Alvarez, P. V. Gratz, D. A. Jiménez, and M. Casas, "Page size aware cache prefetching," in *Proc. 55th IEEE/ACM Int. Symp. Microarchitecture*, 2022, pp. 956–974.

[4] N. Talati et al., "Prodigy: Improving the memory latency of data-indirect irregular workloads using hardware-software co-design," in *Proc. IEEE Int. Symp. High- Perform. Comput. Archit.*, 2021, pp. 654–667.

[5] F. Xue et al., "Tyche: An efficient and general prefetcher for indirect memory accesses," *ACM Trans. Archit. Code Optim.*, vol. 21, no. 2, pp. 1–26, 2024.

[6] B. Panda, "CLIP: Load criticality based data prefetching for bandwidth-constrained many-core systems," in *Proc. 56th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2023, pp. 714–727.

[7] A. V. Jamet, G. Vavouliotis, D. A. Jiménez, L. Alvarez, and M. Casas, "A two level neural approach combining off-chip prediction with adaptive prefetch filtering," in *Proc. 2024 IEEE Int. Symp. High- Perform. Comput. Archit.*, 2024, pp. 528–542.

[8] D. Deb, J. Jose, and M. Palesi, "COPE: Reducing cache pollution and network contention by inter-tile coordinated prefetching in noc-based mpsocs," *ACM Trans. Des. Automat. Electron. Syst.*, vol. 26, no. 3, pp. 1–31, 2020.

[9] B. Panda, "The game of latency, bandwidth, and hardware prefetching," *Computer*, vol. 57, no. 6, pp. 122–126, 2024.

[10] R. Bera et al., "Hermes: Accelerating long-latency load requests via perceptron-based off-chip load prediction," in *Proc. 55th IEEE/ACM Int. Symp. Microarchitecture*, 2022, pp. 1–18.

[11] S. Srinath, O. Mutlu, H. Kim, and Y. N. Patt, "Feedback directed prefetching: Improving the performance and bandwidth-efficiency of hardware prefetchers," in *Proc. IEEE 13th Int. Symp. High Perform. Comput. Archit.*, 2007, pp. 63–74.

[12] N. R. Holtryd, M. Manivannan, P. Stenström, and M. Pericàs, "CBP: Coordinated management of cache partitioning, bandwidth partitioning and prefetch throttling," in *Proc. 30th Int. Conf. Parallel Architectures Compilation Techn.*, 2021, pp. 213–225.

[13] B. Panda, "SPAC: A synergistic prefetcher aggressiveness controller for multi-core systems," *IEEE Trans. Comput.*, vol. 65, no. 12, pp. 3740–3753, Dec. 2016.

[14] M. M. Rafique and Z. Zhu, "Memory-side prefetching scheme incorporating dynamic page mode in 3D-stacked dram," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 11, pp. 2734–2747, Nov. 2021.

[15] C. Navarro, J. Feliu, S. Petit, M. E. Gomez, and J. Sahuquillo, "Bandwidth-aware dynamic prefetch configuration for IBM power8," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 8, pp. 1970–1982, Aug. 2020.

[16] F. Eris, M. Louis, K. Eris, J. Abellan, and A. Joshi, "Puppeteer: A random forest based manager for hardware prefetchers across the memory hierarchy," *ACM Trans. Archit. Code Optim.*, vol. 20, no. 1, pp. 1–25, 2022.

[17] D. Deb and J. Jose, "ZPP: A dynamic technique to eliminate cache pollution in noc based mpsocs," *ACM Trans. Embedded Comput. Syst.*, vol. 22, no. 5s, pp. 1–25, 2023.

[18] S. Jiang, Q. Yang, and Y. Ci, "Merging similar patterns for hardware prefetching," in *Proc. 55th IEEE/ACM Int. Symp. Microarchitecture*, 2022, pp. 1012–1026.

[19] K. Zhou et al., "Improving cache performance for large-scale photo stores via heuristic prefetching scheme," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 9, pp. 2033–2045, Sep. 2019.

[20] G. Ayers, H. Litz, C. Kozyrakis, and P. Ranganathan, "Classifying memory access patterns for prefetching," in *Proc. Twenty-Fifth Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2020, pp. 513–526.

[21] Y. Cui, W. Chen, X. Cheng, and J. Yi, "Hyperion: A highly effective page and pc based delta prefetcher," *ACM Trans. Archit. Code Optim.*, vol. 21, 2024, Art. no. 68.

[22] A. Navarro-Torres, B. Panda, J. Alastruey-Benedé, P. Ibáñez, V. Viñals-Yúfera, and A. Ros, "Berti: An accurate local-delta data prefetcher," in *Proc. 55th IEEE/ACM Int. Symp. Microarchitecture*, 2022, pp. 975–991.

[23] S. Pakalapati and B. Panda, "Bouquet of instruction pointers: Instruction pointer classifier-based spatial hardware prefetching," in *Proc. 2020 ACM/IEEE 47th Annu. Int. Symp. Comput. Archit.*, 2020, pp. 118–131.

[24] L. M. AlBarakat, P. V. Gratz, and D. A. Jiménez, "SLAP-CC: Set-level adaptive prefetching for compressed caches," in *Proc. IEEE 40th Int. Conf. Comput. Des.*, 2022, pp. 50–58.

[25] J. Kim, S. H. Pugsley, P. V. Gratz, A. N. Reddy, C. Wilkerson, and Z. Chishti, "Path confidence based lookahead prefetching," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2016, pp. 1–12.

[26] E. Ebrahimi, O. Mutlu, C. J. Lee, and Y. N. Patt, "Coordinated control of multiple prefetchers in multi-core systems," in *Proc. 42nd Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2009, pp. 316–326.

[27] M. Lurbe, J. Feliu, S. Petit, M. E. Gómez, and J. Sahuquillo, "DeepP: Deep learning multi-program prefetch configuration for the IBM power 8," *IEEE Trans. Comput.*, vol. 71, no. 10, pp. 2646–2658, Oct. 2022.

[28] E. Bhatia, G. Chacon, S. Pugsley, E. Teran, P. V. Gratz, and D. A. Jiménez, "Perceptron-based prefetch filtering," in *Proc. ACM/IEEE 46th Annu. Int. Symp. Comput. Archit.*, 2019, pp. 1–13.

[29] X. Lu, R. Wang, and X.-H. Sun, "Apac: An accurate and adaptive prefetch framework with concurrent memory access analysis," in *Proc. IEEE 38th Int. Conf. Comput. Des.*, 2020, pp. 222–229.
[30] W. Heirman, K. D. Bois, Y. Vandriessche, S. Eyerman, and I. Hur, "Near-side prefetch throttling: Adaptive prefetching for high-performance many-core processors," in *Proc. 27th Int. Conf. Parallel Architectures Compilation Techn.*, 2018, pp. 1–11.
[31] S. Sethumurugan, J. Yin, and J. Sartori, "Designing a cost-effective cache replacement policy using machine learning," in *Proc. 2021 IEEE Int. Symp. High- Perform. Comput. Archit.*, 2021, pp. 291–303.
[32] Q. Duong, A. Jain, and C. Lin, "A new formulation of neural data prefetching," in *Proc. ACM/IEEE 51st Annu. Int. Symp. Comput. Archit.*, 2024, pp. 1173–1187.
[33] S. Zangeneh, S. Pruett, S. Lym, and Y. N. Patt, "Branchnet: A convolutional neural network to predict hard-to-predict branches," in *Proc. 53 rd Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2020, pp. 118–130.
[34] E. Garza, S. Mirbagher-Ajorpaz, T. A. Khan, and D. A. Jiménez, "Bit-level perceptron prediction for indirect branches," in *Proc. ACM/IEEE 46th Annu. Int. Symp. Comput. Archit.*, 2019, pp. 27–38.
[35] L. Peled, U. Weiser, and Y. Etsion, "A neural network prefetcher for arbitrary memory access patterns," *ACM Trans. Archit. Code Optim.*, vol. 16, no. 4, pp. 1–27, 2019.
[36] P. Zhang, A. Srivastava, B. Brooks, R. Kannan, and V. K. Prasanna, "Raop: Recurrent neural network augmented offset prefetcher," in *Proc. Int. Symp. Memory Syst.*, 2020, pp. 352–362.
[37] L. Jia, J. P. Mcmahon, S. Gudaparthi, S. Singh, and R. Balasubramonian, "Pathfinder: Practical real-time learning for data prefetching," in *Proc. 29th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2024, pp. 785–800.
[38] Z. Shi, A. Jain, K. Swersky, M. Hashemi, P. Ranganathan, and C. Lin, "A hierarchical neural model of data prefetching," in *Proc. 26th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.*, 2021, pp. 861–873.
[39] R. Jain, P. R. Panda, and S. Subramoney, "Cooperative multi-agent reinforcement learning-based co-optimization of cores, caches, and on-chip network," *ACM Trans. Archit. Code Optim.*, vol. 14, no. 4, pp. 1–25, 2017.
[40] G. Modi, A. Bagchi, N. Jindal, A. Mandal, and P. R. Panda, "Cabarre: Request response arbitration for shared cache management," *ACM Trans. Embedded Comput. Syst.*, vol. 22, no. 5s, pp. 1–24, 2023.
[41] Y. Wang, W. Zhang, M. Hao, and Z. Wang, "Online power management for multi-cores: A reinforcement learning based approach," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 4, pp. 751–764, Apr. 2022.
[42] B. Donyanavard et al., "SOSA: Self-optimizing learning with self-adaptive control for hierarchical system-on-chip management," in *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2019, pp. 685–698.
[43] X. Chen, J. Hu, Z. Chen, B. Lin, N. Xiong, and G. Min, "A reinforcement learning-empowered feedback control system for industrial Internet of Things," *IEEE Trans. Ind. Inform.*, vol. 18, no. 4, pp. 2724–2733, Apr. 2022.
[44] R. Bera, K. Kanellopoulos, A. Nori, T. Shahroodi, S. Subramoney, and O. Mutlu, "Pythia: A customizable hardware prefetching framework using online reinforcement learning," in *Proc. 54th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2021, pp. 1121–1137.
[45] S. Padakandla, "A survey of reinforcement learning algorithms for dynamically varying environments," *ACM Comput. Surv.*, vol. 54, no. 6, pp. 1–25, 2021.
[46] A. K. Shakya, G. Pillai, and S. Chakrabarty, "Reinforcement learning algorithms: A brief survey," *Expert Syst. Appl.*, vol. 231, 2023, Art. no. 120495.
[47] R. Bera, A. V. Nori, O. Mutlu, and S. Subramoney, "Dspatch: Dual spatial pattern prefetcher," in *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2019, pp. 531–544.
[48] G. Sun, J. Shen, and A. V. Veidenbaum, "Combining prefetch control and cache partitioning to improve multicore performance," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2019, pp. 953–962.
[49] N. Gober et al., "The championship simulator: Architectural simulation for education and competition," 2022, *arXiv:2210.14324*.
[50] G. Gerogiannis and J. Torrellas, "Micro-armed bandit: Lightweight & reusable reinforcement learning for microarchitecture decision-making," in *Proc. 56th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2023, pp. 698–713.
[51] J. Doweck, "White paper inside intel core microarchitecture and smart memory access," *Intel Corporation*, vol. 52, pp. 72–87, 2006.
[52] "SPEC CPU 2017,," 2017. [Online]. Available: https://www.spec.org/cpu2017/
[53] M. He et al., "DSDP: Dual stream data prefetcher," in *Proc. Int. Conf. Parallel Architectures Compilation Techn.*, 2022, pp. 372–383.
[54] "SPEC CPU 2006," 2006. [Online]. Available: https://www.spec.org/cpu2006/
[55] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proc. 17th Int. Conf. Parallel Architectures Compilation Techn.*, 2008, pp. 72–81.
[56] J. Shun and G. E. Blelloch, "Ligra: A lightweight graph processing framework for shared memory," in *Proc. 18th ACM SIGPLAN Symp. Princ. Pract. Parallel Program.*, 2013, pp. 135–146.
[57] M. Ferdman et al., "Clearing the clouds: A study of emerging scale-out workloads on modern hardware," *ACM SIGPLAN Notices*, vol. 47, no. 4, pp. 37–48, 2012.

Huijing Yang (Student Member, IEEE) received the BS degree from Zhoukou Normal University, Zhoukou, China in 2018. And she is currently working toward the PhD degree with the Beijing University of Technology, Beijing, China. She is a student member of CCF. Her main research direction is computer architecture.

Juan Fang (Senior Member, IEEE) received the MS degree from the Jilin University of Technology, Changchun, China, in 1997, and the PhD degree from the College of Computer Science, Beijing University of Technology, Beijing, China, in 2005. In 1997, she joined the College of Computer Science, Beijing University of Technology. From 2015, she is the professor with the Beijing University of Technology. Her research interests include high performance computing, edge computing, and Big Data technology.

Yumin Hou received the BS degree in microelectronics from the University of Electronic Science and Technology of China, Chengdu, China, in 2012, and the PhD degree in electronics science and technology from Tsinghua University, Beijing, China, in 2018. From 2019 to 2021, she was a postdoc with the University of Florida. She is currently a lecturer with the Faculty of Information Technology, Beijing University of Technology, Beijing, China. Her research interests include computer architecture, processing-in-memory architecture, and hardware security.

Xing Su received the BSc degree from the School of Software Engineering, Beijing University of Technology, in 2007, and the MSc and PhD degrees in computer science from the University of Wollongong, Australia, in 2012 and 2015. He an associate professor with the Faculty of Information Technology, Beijing University of Technology, China. His research interests include distributed artificial intelligence, multi-agent systems, disaster management, and service-oriented computing.

Neal N. Xiong (Senior Member, IEEE) received the PhD degree in sensor system engineering from Wuhan University, in 2007, and the PhD degree in dependable communication networks from the Japan Advanced Institute of Science and Technology, in 2008. He is currently a Computer Science Program Chair and an associate professor with the Department of Computer Science and Mathematics, Sul Ross State University, Alpine, TX 79830, USA. Before joining Sul Ross State University, he worked at Georgia State University, Northeastern State University, and Colorado Technical University (Full professor for about 4 years) for around 14 years. His research interests include cloud computing, security and dependability, AI, data analysis, parallel and distributed computing, and networks.