A Survey on Data Prefetching Techniques for Performance Optimization in Modern Computing Architectures

www.surveyx.cn

Abstract

Data prefetching is a critical technique in modern computing architectures, addressing the Memory Wall problem by reducing memory access latency and enhancing processor efficiency. This survey explores the significance of data prefetching across various domains, including high-performance computing, cloud environments, and serverless workflows. It highlights the role of prefetching in optimizing cache memory usage, reducing power consumption, and improving performance in data-centric applications. The survey identifies the challenges of traditional prefetching methods, particularly in adapting to multifarious memory access patterns and the trade-off between cache miss coverage and accuracy. It proposes innovative prefetching techniques, such as the Multi-Lookahead Offset Prefetcher (MLOP), to optimize miss coverage and timeliness in data cache management. The survey also emphasizes the importance of runtime mechanisms, hardware architecture, and performance optimization strategies in enhancing prefetching effectiveness. It underscores the need for hardware-software co-design and advanced cache management techniques to address the evolving demands of modern computing environments. The survey concludes by identifying future research directions, including the exploration of heterogeneous precision assignments and the integration of advanced models to further improve prefetching performance and energy efficiency.

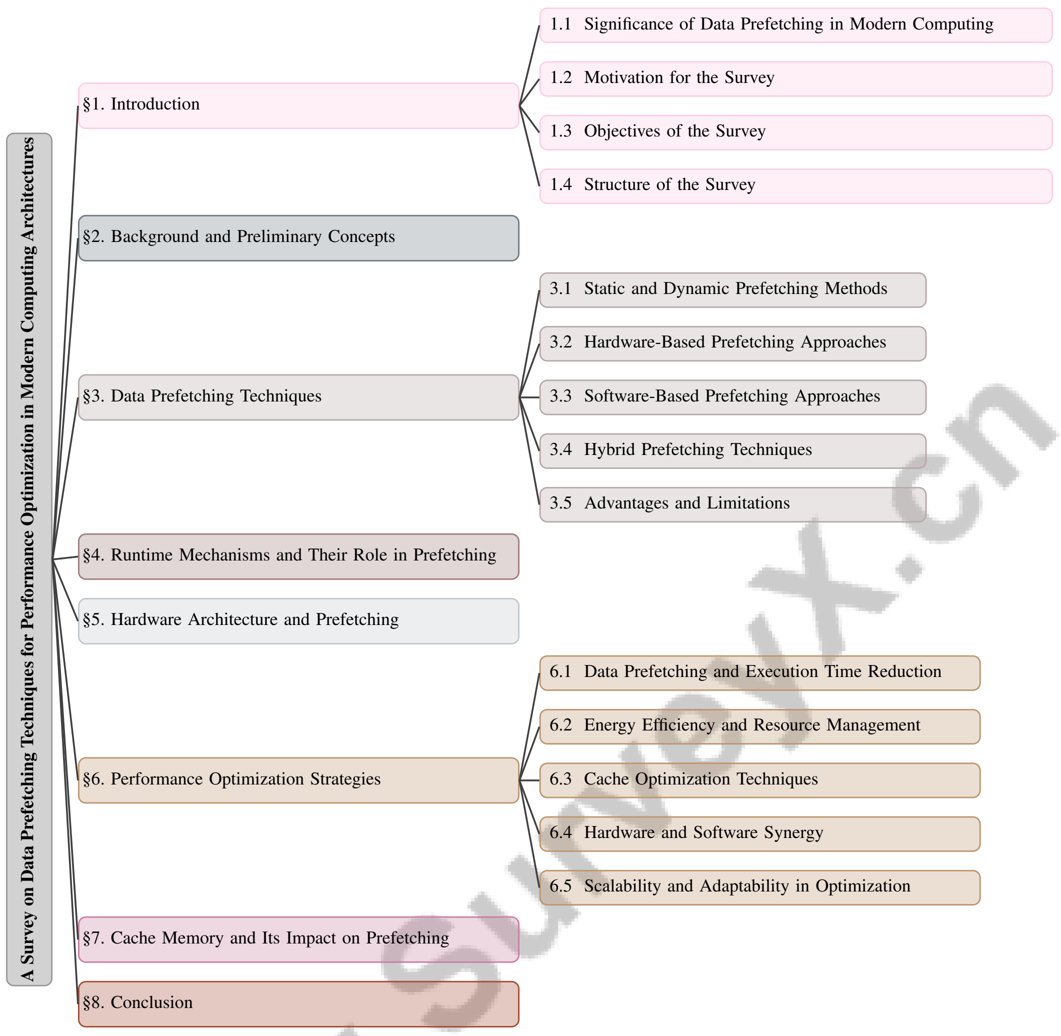
1 Introduction

1.1 Significance of Data Prefetching in Modern Computing

Data prefetching is essential in modern computing architectures, significantly improving performance by mitigating the Memory Wall problem, which arises from the disparity between processor speeds and memory access times. By preloading data into cache memory prior to processor requests, prefetching reduces memory access latency and sustains high processor efficiency and throughput, particularly in high-performance computing and data analytics environments where memory bandwidth is often a critical bottleneck [1].

In cloud computing, unique memory access patterns diverge from traditional workloads, resulting in inefficiencies in existing prefetching schemes [2]. Adapting prefetching to these specific patterns is vital for optimizing workloads. In data-intensive serverless workflows, prefetching alleviates challenges related to cold starts and inter-function data passing, enhancing application performance [3].

Moreover, prefetching is crucial for addressing power consumption in application-specific instruction set processor (ASIP) design methodologies. By improving computing performance, prefetching contributes to energy-efficient systems [4]. In web applications, prefetching techniques are necessary for performance enhancement amid the growing size of web pages and the inefficiencies in resource delivery [5].


A Survey on Data Prefetching Techniques for Performance Optimization in Modern Computing Architectures
§1. Introduction
1.1 Significance of Data Prefetching in Modern Computing
1.2 Motivation for the Survey
1.3 Objectives of the Survey
1.4 Structure of the Survey
§2. Background and Preliminary Concepts
§3. Data Prefetching Techniques
3.1 Static and Dynamic Prefetching Methods
3.2 Hardware-Based Prefetching Approaches
3.3 Software-Based Prefetching Approaches
3.4 Hybrid Prefetching Techniques
3.5 Advantages and Limitations
§4. Runtime Mechanisms and Their Role in Prefetching
§5. Hardware Architecture and Prefetching
§6. Performance Optimization Strategies
6.1 Data Prefetching and Execution Time Reduction
6.2 Energy Efficiency and Resource Management
6.3 Cache Optimization Techniques
6.4 Hardware and Software Synergy
6.5 Scalability and Adaptability in Optimization
§7. Cache Memory and Its Impact on Prefetching
§8. Conclusion
Figure 1: chapter structure

In many-core processor systems, an increase in concurrent threads can lead to competitive cache misses. Prefetching mitigates this issue by ensuring readily available frequently accessed data [6], which is vital in data-centric applications where high data parallelism is often underutilized by conventional processors [7]. Additionally, in interactive data visualization and exploration applications, prefetching addresses network bottlenecks and bursty request patterns, enhancing user experience by reducing latency [8].

The significance of prefetching extends to optimizing cache-aided networks, where cache memory size and update rates are crucial for performance enhancement [9]. Advanced techniques, such as mixed-precision embedding training utilizing high-precision caches, exemplify the transformative impact of prefetching in achieving substantial memory savings while maintaining performance accuracy [10].

In distributed context management systems, prefetching supports adaptive caching strategies based on performance metrics, overcoming limitations of existing methods and improving system adaptability [11]. Multi-objective approaches for optimizing cache configurations are essential for balancing performance and energy consumption, particularly in multimedia applications [12].

The importance of prefetching is further highlighted in stencil computations on regular lattices, where traditional methods lack performance models for estimating speedup, necessitating innovative optimization approaches [13]. In hardware prefetching, the trade-off between coverage and accuracy critically influences the effectiveness of prefetching mechanisms [14]. The increasing complexity of workloads and the recent slowdown in Moore's law scaling necessitate new approaches for efficient computing, with data prefetching serving as a vital component in addressing these challenges [15].

1.2 Motivation for the Survey

This survey is motivated by the urgent need to address the inefficiencies of traditional data prefetching methods, which are becoming inadequate for modern computing architectures advancing toward the Exascale era [16]. The complexity of diverse memory access patterns in hybrid applications exacerbates these challenges, resulting in suboptimal performance [17]. Current strategies often struggle with the trade-off between high cache miss coverage and prefetching accuracy, leading to inefficiencies [14]. This is notably evident in GPU-based operations, such as Sparse Matrix-Vector Multiplication (SpMV), where optimizing data movement costs remains a significant challenge [18].

In high-performance computing, understanding data movement and reuse is critical for effective performance tuning [19]. While machine learning applications in software optimizations are advancing, their application in computer hardware architecture remains largely unexplored [15]. The inefficiencies and complexities associated with off-chip metadata in temporal prefetching hinder performance and increase energy consumption [20], necessitating a reevaluation of these methods [21]. In IoT environments, memory object caching systems often lack prefetch or read-ahead functions, which are crucial for effectively managing IoT queries [22].

The inference phase of Large Language Models (LLMs) presents challenges, as excessive memory consumption during long-context inputs can lead to out-of-memory errors [23]. Furthermore, simple prefetching schemes have proven inefficient for cloud workloads, highlighting the need for innovative approaches to enhance performance efficiency in last-level caches and off-chip memory for cloud applications [2]. The increasing size and complexity of web applications also contribute to performance bottlenecks and user experience degradation [5].

Additionally, the high energy consumption in instruction caches of extensible processors, particularly how ISA extensions influence cache memory access, is a significant concern [4]. Network traffic congestion, driven by the growing number of users and demand for high data rate content, further emphasizes the relevance of this survey in addressing current computing challenges [24]. The need for improved prefetching methods that effectively balance timeliness and miss coverage is crucial [25], as is the motivation for enhancing programmability, performance portability, and resource efficiency in complex architectures due to existing narrow hardware-software interfaces [26].

This survey aims to provide a comprehensive overview of these challenges and propose solutions that enhance the adaptability and efficiency of prefetching mechanisms in modern computing environments, particularly in serverless function execution, where reliance on external storage services introduces latency [3]. Additionally, it seeks to address challenges in optimizing delivery rates in cache networks, especially when the number of users exceeds the number of files [27].

1.3 Objectives of the Survey

The primary objective of this survey is to explore innovative data prefetching techniques and their substantial impact on performance optimization within modern computing architectures. By investigating advanced strategies, such as the Multi-Lookahead Offset Prefetcher (MLOP), the survey aims to optimize both miss coverage and timeliness in data cache management, thereby addressing critical performance bottlenecks associated with memory access patterns in contemporary workloads [25]. Furthermore, the survey seeks to enhance the scalability and accuracy of runtime performance predictions across diverse hardware configurations through novel approaches, as highlighted in recent research [28].

A significant goal of the survey is to propose novel caching solutions, including those that leverage prefetching mechanisms to improve system performance and energy efficiency. This includes examining methods for customizing cache sizes for each layer based on the importance of cached data, leading to significant memory savings while maintaining accuracy. Additionally, the survey intends to explore strategies for optimizing data delivery rates in cache networks, particularly during peak traffic periods, to efficiently satisfy all user demands [27].

The survey also aims to address the inefficiencies in resource management and performance optimization caused by ineffective communication between application-level semantics and hardware. By enhancing programmability, performance portability, and resource efficiency, the survey endeavors to advance the field of data prefetching, ensuring that the techniques are adaptable and efficient across diverse computing environments [26].

The survey aims to significantly enhance system performance, energy efficiency, and scalability in modern computing environments by evaluating various hardware data prefetching techniques. By focusing on the latest advancements and methodologies in prefetching, the survey ensures that these techniques continue to lead performance optimization strategies, particularly in the context of increasingly complex server workloads and evolving computing architectures. This contribution is crucial for addressing the memory access bottlenecks that hinder the performance of high-demand applications in data centers and cloud computing infrastructures [29, 30, 31, 14].

1.4 Structure of the Survey

This survey is meticulously structured to provide a comprehensive exploration of data prefetching techniques and their role in performance optimization within modern computing architectures. The paper opens with a comprehensive that highlights the critical role of data prefetching in modern computing environments, particularly in the context of high-performance server workloads that demand low latency and quality-of-service. It articulates the motivation for this survey by addressing the increasing complexity of data access patterns in large-scale data centers and the need for efficient memory management strategies. Additionally, the introduction specifies the primary objectives of the exploration, which include examining recent advancements in hardware data prefetching techniques and their implications for optimizing performance in contemporary applications [29, 30, 25, 14].

The subsequent section, **Background and Preliminary Concepts**, delves into key definitions and concepts pertinent to data prefetching, runtime mechanisms, hardware architecture, and performance optimization. This section aims to elucidate the interconnections between these concepts and their collective importance in enhancing computing performance.

In the **Data Prefetching Techniques** section, the survey examines a variety of prefetching methods, including static and dynamic approaches, hardware-based and software-based strategies, and hybrid techniques that combine both elements. The advantages and limitations of these methods are critically analyzed to provide a nuanced understanding of their applicability in different computing contexts.

The role of **Runtime Mechanisms in Prefetching** is explored in-depth, focusing on how runtime operations such as memory management and execution flow influence the effectiveness of prefetching. This section also investigates the impact of parallelization on enhancing runtime mechanisms.

The survey then transitions to an analysis of **Hardware Architecture and Prefetching**, assessing how different CPU designs, memory hierarchies, and GPU architectures affect prefetching strategies. The exploration of hardware-software co-design for optimizing data prefetching techniques highlights its potential to enhance performance in memory-intensive applications, particularly in server environments where latency and bandwidth constraints are critical. This approach aims to improve prefetching accuracy and coverage while addressing the challenges of aggressive prefetching, such as cache evictions and bandwidth saturation, by leveraging adaptive mechanisms and advanced predictors to fine-tune prefetching strategies effectively [29, 32, 31, 33, 14].

In the **Performance Optimization Strategies** section, various techniques that incorporate data prefetching are discussed, with an emphasis on execution time reduction, energy efficiency, and cache optimization. The synergy between hardware and software in achieving performance gains and the scalability of optimization strategies are also examined.

The investigation into the role of reveals that factors such as cache size, speed, and architecture significantly influence prefetching efficiency and overall system performance. Specifically, enhancements in prefetching techniques, such as translation-enabled memory prefetching and perceptron-based filtering, demonstrate that optimizing cache interactions can lead to substantial improvements in memory access times and resource utilization, thereby boosting system throughput by 10-30% and reducing memory request counts by up to 83.84% without compromising instruction execution rates. These findings underscore the critical importance of cache design in addressing memory bottlenecks in modern computing environments [20, 34, 25, 31, 14]. Innovative cache management techniques and methods for evaluating cache performance are also discussed.

In the **Conclusion**, the survey encapsulates the pivotal findings regarding data prefetching, highlighting its essential role in enhancing system performance by effectively predicting and loading future memory accesses to mitigate latency issues. The authors underscore the widespread implementation of various data prefetching techniques in high-performance processors, which cater to the demands of server workloads across large-scale data centers. Furthermore, the conclusion identifies promising avenues for future research, including the development of scheduling-aware mechanisms that optimize data management in cloud environments, ensuring efficient resource utilization without compromising the performance of concurrent applications [29, 30].The following sections are organized as shown in Figure 1.

2 Background and Preliminary Concepts

2.1 Definitions and Key Concepts

Data prefetching, a proactive strategy to load data blocks into memory before processor requests, mitigates memory latency and enhances performance [30]. This is crucial for addressing the memory wall problem, where the speed disparity between processors and memory access creates bottlenecks, particularly in in-order processors susceptible to stalls from data dependencies [35]. Cache memory, specifically the Last-Level Cache (LLC), plays a vital role in performance by storing frequently accessed data close to the CPU, thus reducing access times [36]. Advanced algorithms like the Adaptive Weight Ranking Policy (AWRP) improve cache efficiency by ranking pages based on recency and frequency [37].

Performance optimization focuses on enhancing execution efficiency, a complex task given the need to optimize deep learning workloads influenced by cache hierarchies and SIMD architectures [38]. The Execution-Cache-Memory (ECM) model and the Roofline Model are essential frameworks for analyzing performance by examining instruction throughput, data transfers, and balancing computational intensity with memory bandwidth [39, 40].

In heterogeneous computing, characterizing memory access patterns, especially in OpenCL workloads, is critical for performance [41]. The dynamic evaluation framework (DEF) aids in simulating access patterns to assess adaptability [42]. Accurate energy consumption and throughput estimation in cache architectures are vital for optimizing multicore systems during design and runtime [43].

Unified Virtual Memory (UVM) facilitates on-demand data movement between CPU and GPU memory but suffers from bottlenecks due to high latency from page table walks and data migration [44]. Current prefetching methods in UVM often fail to predict memory access patterns accurately, leading to performance degradation [44]. Effective prefetching can mitigate cache misses, yet challenges remain in determining optimal prefetching strategies [45].

Neural Prefetching, using machine learning to predict memory access based on historical patterns, represents a significant advancement in prefetching strategies [15]. This approach is beneficial for optimizing memory-rate tradeoffs in multiuser caching systems [46]. Delta prefetching methods, utilizing global deltas to anticipate data needs, exemplify innovation in this area [47]. Balancing memory usage and communication load during file delivery is critical in caching systems, especially when a server with limited files serves multiple users with local caches [48].

These foundational concepts underscore the importance of data prefetching and performance optimization in modern computing, emphasizing cache structures, memory latency, and bandwidth in enhancing system efficiency. The complexity of cache analysis is pronounced in safety-critical applications where bounding worst-case execution times is crucial [49]. The challenges of implementing optimization techniques on large-scale supercomputers, characterized by hardware heterogeneity and complex memory access patterns, highlight the ongoing need for advancements [16]. The design of high-speed, low-power, and non-volatile memory for on-chip cache applications illustrates the evolving landscape of cache technology [50].

Key concepts such as performance modeling tools, runtime performance prediction, and the complexity of modern hardware architectures are integral to understanding performance optimization [28]. Additionally, programmability, performance portability, resource efficiency, and cross-layer abstractions are vital for enhancing prefetching methodologies' adaptability and efficiency [26].

2.2 Interrelation of Concepts

The interplay among data prefetching, runtime mechanisms, hardware architecture, performance optimization, and cache memory is foundational for enhancing computing performance. Data prefetching bridges the latency gap between processor speeds and memory access times, particularly in multi-core systems using shared cache models, optimizing execution time and improving overall system efficiency [51]. The complexity of cache behavior analysis, especially with various replacement policies like LRU, FIFO, and PLRU, underscores the necessity for effective prefetching strategies [49].

Runtime mechanisms related to memory management are crucial for successful data prefetching, enabling dynamic adaptation to workloads and access patterns, optimizing resource utilization, and minimizing latency. The inefficiency of current software runtimes, due to reliance on virtual memory mechanisms that introduce high overhead, highlights the need for advanced prefetching strategies [52]. PagedAttention, for example, allows key-value caches in non-contiguous memory blocks, reducing waste [53].

Hardware architecture, including CPU designs and memory hierarchies, directly influences prefetching techniques' efficacy. Scalable reuse distance-based shared memory models predicting cache hit rates exemplify the importance of considering hardware characteristics in performance evaluations [54]. This consideration is critical for addressing performance bottlenecks related to memory bandwidth and instruction throughput, often overlooked in conventional benchmarks [39]. The interrelation between processor architectures and application demands further emphasizes adapting architectures to specific memory access patterns for efficient resource utilization [55].

Performance optimization strategies are closely linked to effective cache memory management, essential for reducing data access times. The challenge lies in organizing cache memory to maximize utilization while minimizing access time when fetching pages from main memory [56]. AdaptSize, using a Markov cache model that dynamically tunes caching parameters, highlights the necessity for dynamic cache management approaches [57]. Existing methods often perform optimally in high cache memory regimes or are limited in low cache contexts, creating a gap in achieving optimal performance across all scenarios [46].

Lastly, the efficiency of optimization techniques depends on accurately evaluating database performance in the context of changing access patterns. This evaluation is essential for understanding the interrelation of concepts and ensuring prefetching strategies are adaptable to diverse computing environments. Recent research has advanced understanding of web performance, leading to practical techniques that enhance user experiences and reduce resource wastage [5].

The interconnected concepts of data prefetching underscore its critical role in enhancing performance in contemporary computing environments. This highlights the necessity for comprehensive strategies that address diverse challenges associated with optimizing performance, particularly in high-demand applications like cloud computing and data-intensive scientific workflows. Strategies must consider factors like memory access efficiency, cache resource management, and adaptation to multi-tiered storage systems to ensure data is readily available for processing while minimizing latency and maximizing resource utilization [29, 30, 58, 14]. The intricate relationship between software execution and hardware performance emphasizes the complexity of constructing analytic performance models, necessitating innovative approaches to prefetching and cache management.

3 Data Prefetching Techniques

Data prefetching techniques are integral in enhancing data access and processing efficiency within computing systems. Table 1 presents a detailed categorization of data prefetching techniques, illustrating the diverse methodologies employed to enhance data access efficiency and computational performance. Table 4 offers a comprehensive comparison of data prefetching methods, elucidating their distinct characteristics and optimization strategies across different computing environments. This section delves into the primary categories of data prefetching, focusing on static and dynamic methods. These strategies cater to varying application requirements and memory access patterns, optimizing performance and resource utilization in modern computing environments.

Category	Feature	Method
Static and Dynamic Prefetching Methods	Pattern and Locality Utilization	ReRe[55], PSL[41]
	Static Analysis and Prediction	SADP[30], Kraft[59]
	Dynamic Prefetching Optimization	KH[8], NST[33]
	Caching Strategies	CCCS[24]
Hardware-Based Prefetching Approaches	Predictive Prefetching	ADRA[60], ARPA[1], GPGPU-COT[61], CLIP[32]
	Non-Volatile Caching	RRAP[62]
Software-Based Prefetching Approaches	Model-Driven Techniques	DLPP[44], ReVeLA[63]
	Pattern-Based Prediction	Bingo[64], Domino[21]
Hybrid Prefetching Techniques	Adaptive Techniques	CDOS[65], RPG2[45]
	Cache-Based Strategies	BRT[47], CC[7], TEMPO[31]
	Hierarchical Structures	HCS[56]
Advantages and Limitations	Memory and Data Management	EHYB[18], POM-FGIC[66], NP[15], Triage[20], Kona[52], XMem[26]
	Prefetching and Caching Optimization	SMC[22], CCCS[27], ITC[9], ACOCA[11], PPF[14], XKV[23], MLOP[25]
	Performance and Efficiency Enhancement	HF[58], MOEACO[12], ECM[13], CSR-ISA[4]

Table 1: This table provides a comprehensive overview of various data prefetching techniques, categorized into static and dynamic methods, hardware-based approaches, software-based strategies, and hybrid techniques. It highlights the specific features and methods employed within each category, offering detailed insights into the tools and methodologies that optimize data access and improve computational performance. The table also discusses the advantages and limitations of these techniques, emphasizing their impact on memory management and system efficiency.

3.1 Static and Dynamic Prefetching Methods

Static and dynamic prefetching methods play crucial roles in optimizing data access by tailoring strategies to specific application needs and memory access patterns. As illustrated in Figure 2, prefetching methods can be categorized into static, dynamic, and hybrid approaches, highlighting key tools and techniques that optimize data access efficiency. Static prefetching, effective for predictable access patterns like those in scientific computing, relies on compile-time analysis to schedule data loads [67]. Tools such as Kerncraft are used for static source code analysis, optimizing prefetching decisions based on loop kernel metrics [59]. Near-Side Prefetch Throttling (NST) refines static prefetching by adjusting prefetch distance, reducing unnecessary data loads [33].

Dynamic prefetching adapts to runtime conditions, making real-time decisions based on observed memory access patterns. Frameworks like Khameleon optimize response quality and latency through progressive encoding and server-side scheduling [8]. Dynamic prefetching is particularly effective for irregular memory access behaviors, such as in graph processing applications, where adaptive learning can enhance data access efficiency [1]. The SADP technique, which predicts data block access based on job scheduling, exemplifies a sophisticated dynamic prefetching strategy [30].

Innovations in dynamic prefetching include Berti, a first-level data cache prefetcher that uses local deltas from demand accesses to anticipate future data needs, reducing cache misses [47]. The Recursive Reduce (ReRe) methodology identifies essential memory access patterns, informing both static and dynamic prefetching methods [55]. The Parallel Spatial Locality Metric (PSL) provides insights into optimizing prefetching strategies for parallel computing environments [41]. The Explicit Caching Hybrid (EHYB) framework minimizes data movement in GPU-based Sparse Matrix-Vector Multiplication (SpMV) by partitioning and caching input vectors in shared memory [18].

The choice between static and dynamic prefetching depends on application requirements and workload characteristics. Static prefetching suits regular access patterns, while dynamic prefetching offers flexibility in unpredictable environments. Both methods enhance performance and resource efficiency, underscoring the importance of selecting appropriate strategies based on application needs. Reinforcement learning-based approaches, such as ACOCA, further enhance dynamic prefetching by adaptively managing context caching [11]. The Centralized Coded Caching Scheme (CCCS) employs a two-phase approach for caching and delivering content, bridging static and dynamic methods [24]. The MLOP method evaluates multiple offsets and lookahead levels, contrasting with traditional methods relying on a single best offset [25].

3.2 Hardware-Based Prefetching Approaches

Hardware-based prefetching techniques enhance system performance by leveraging specific hardware features to anticipate and load data into cache memory before processor requests. These techniques

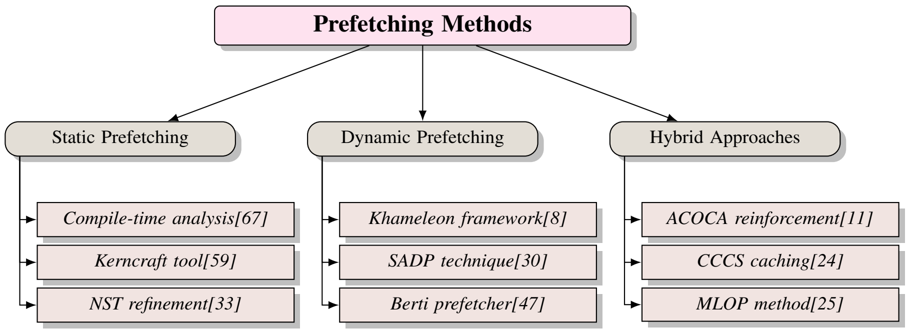


Figure 2: This figure illustrates the categorization of prefetching methods into static, dynamic, and hybrid approaches, highlighting key tools and techniques that optimize data access efficiency.

address memory latency and bandwidth limitations in many-core systems. CPU cache optimization strategies, such as blocking and loop fusion, significantly improve GPGPU performance by optimizing data locality [61].

Data Direct I/O (DDIO) technology allows I/O devices to access the Last Level Cache (LLC) directly, reducing latency and improving data throughput [68]. The Read Reference Activity Persistent (RRAP) strategy uses STT-RAM to store frequently accessed cache blocks, optimizing L2 cache performance by reducing read miss ratios [62].

Intelligent prefetching models further enhance hardware-based strategies. Zhang et al.'s dual prefetching scheme combines a fast-response model for regular patterns with an adaptive machine-learning prefetch engine, balancing quick data access with intelligent predictions [1]. The Critical Load Identification and Prefetch (CLIP) mechanism identifies and prefetches load addresses critical for system performance, enhancing data availability in bandwidth-constrained environments [32].

Challenges in hardware-based techniques include the need for precise prediction models to avoid cache pollution. The Perceptron Prefetch Filter (PPF) uses an online-trained neural model to filter inaccurate prefetches, allowing for deeper speculation and higher coverage [14]. The Adaptive Dynamic Replacement Approach (ADRA) combines LRU and LFU principles to efficiently manage cache storage [60].

3.3 Software-Based Prefetching Approaches

Software-based prefetching techniques enhance data access efficiency by leveraging program semantics and runtime information to anticipate future data requirements. These methods are effective where hardware-based solutions may not suffice. The Register Vector Length Agnostic (ReVeLA) prefetcher uses vectorized code semantics to improve prefetch request accuracy [63].

Deep Learning based Page Prefetching (DLPP) employs a Transformer model to predict memory access patterns in CPU-GPU Unified Virtual Memory (UVM) environments, enhancing prefetching accuracy and reducing latency [44]. The Bingo prefetcher associates observed spatial patterns with events to increase prediction accuracy in applications with spatial locality [64]. Domino prefetching uses recent miss patterns to anticipate future data needs, reducing cache misses [21].

Coded prefetching schemes enhance content delivery efficiency, achieving improved memory-rate tradeoff points in multiuser caching systems [24, 46, 27, 69]. These schemes maximize delivery rates while managing limited cache capacities, crucial in networked environments.

Software-based prefetching offers flexibility and adaptability, allowing for tailored strategies dynamically adjusted based on application requirements. These techniques are cost-effective where hardware modifications are impractical, enhancing data access efficiency. Intelligent software prefetching models, such as Perceptron-based Prefetch Filtering, improve coverage and accuracy, meeting the complex demands of modern computing environments [34, 14].

Method Name	Methodology Integration	Dynamic Adaptation	Performance Optimization
CC[7]	Hardware And Software	Real-time Data	Reduce Latency
HF[58]	Server-push Model	Global View	Reduce Cache Pollution
RPG2[45]	Pure-software System	Dynamic Prefetching Adjustments	Optimize Data Prefetching
TEMPO[31]	Hardware Mechanism	Adjusts Strategies	Reduce Latency
HCS[56]	Hybrid Strategies	Random Access Patterns	Improve Cache Utilization
CDOS[65]	Hardware-software Co-design	Dynamic Allocation	Improved Computational Performance
BRT[47]	Hardware And Software	Real-time Data	Reduce Latency

Table 2: Overview of various hybrid prefetching techniques and their integration methodologies, dynamic adaptation strategies, and performance optimization goals. The table compares different methods, highlighting their unique approaches to combining hardware and software solutions to enhance data access efficiency and reduce latency.

3.4 Hybrid Prefetching Techniques

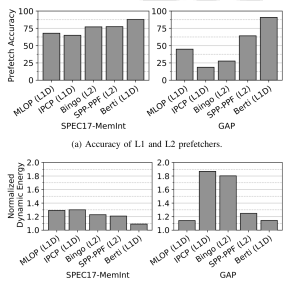
Hybrid prefetching techniques combine hardware and software methodologies to optimize data access and enhance system performance. These techniques leverage the strengths of memory and storage systems, minimizing latency across computing environments. Strategies like Scheduling-Aware Data Prefetching (SADP) and hierarchical data prefetching optimize data placement in low-latency storage and manage memory resources intelligently [56, 29, 58, 30, 31].

The Compute Cache architecture transforms traditional cache elements into computational units, reducing data movement overhead and enhancing prefetching efficiency [7]. The HFetch prefetcher uses a hierarchical design to optimize read operations, dynamically adjusting strategies based on data access patterns [58]. ReSemble integrates multiple prefetchers, selecting the most effective strategy through online learning [17]. RPG 2 adapts prefetching during program execution, enhancing effectiveness in real-time scenarios [45].

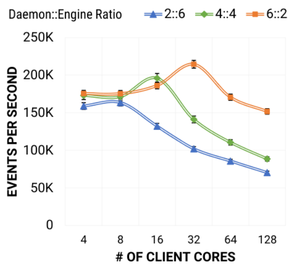
The TEMPO framework leverages memory accesses for page table lookups to prefetch data into the DRAM row buffer and last-level cache, reducing subsequent access latency [31]. New cache strategies optimize page placement, improving data access times and system efficiency [56].

Hybrid techniques offer comprehensive solutions to modern computing challenges by combining hardware and software methodologies. These approaches enable precise data management, reduced latency, and improved system throughput. The dynamic operating system proposed by Schirmer et al. exemplifies hybrid systems managing multiple processing units, enhancing performance through integrated prefetching strategies [65].

Table 2 presents a comprehensive comparison of hybrid prefetching techniques, detailing their methodologies, dynamic adaptation capabilities, and performance optimization strategies.

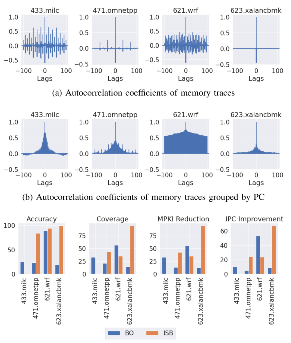


(a) Comparison of L1 and L2 Prefetchers and Energy Consumption in SPEC17-MemInt and GAP Applications[47]



(a) HFecth server-to-client ratio

(b) Performance of a Distributed System with Different Daemon::Engine Ratio Configurations[58]



(c) Autocorrelation Coefficients of Memory Traces and Grouped by PC[17]

Figure 3: Examples of Hybrid Prefetching Techniques

As shown in Figure 3, hybrid prefetching techniques enhance system performance and energy efficiency. The figure illustrates three examples: L1 and L2 prefetcher accuracy and energy consumption in SPEC17-MemInt and GAP applications; distributed system performance with varying Daemon::Engine ratio configurations; and autocorrelation coefficients of memory traces, offering insights into memory access patterns. These examples underscore the complexity and multifaceted nature of hybrid prefetching techniques, emphasizing the need for tailored strategies to optimize both performance and energy usage [47, 58, 17].

3.5 Advantages and Limitations

Method Name	Optimization Techniques	Memory Management	Performance Challenges
PPF[14]	Perceptron-based Filtering	Optimizing Memory Resource	Balancing Prefetching Aggressiveness
HF[58]	Server-push Approach	Cache Pollution Reduction	Increased Latencies
EHYB[18]	Explicitly Caching Input	Shared Memory	Memory-bound Nature
ACOCA[11]	Reinforcement Learning	Selective Caching Strategy	Dynamic Nature
NP[15]	Neural Prefetching	Memory Access Patterns	Prediction Accuracy
ITC[9]	Reducing Multiplicative Gap	Allocate Cache Memory	Inefficient Cache Memory
Triage[20]	Dynamic Cache Partitioning	On-chip Metadata	Balancing Prefetching Aggressiveness
Kona[52]	Cache Coherence Protocols	Cache-line Granularity	High Latency
SMC[22]	Smart Read-ahead	Prefetching Mechanism	Performance Bottlenecks
XKV[23]	Dynamic KV Allocation	Personalized KV Cache	Computational Overhead
CSR-ISA[4]	Cache Size Reduction	Custom Instructions	Balancing Prefetching Aggressiveness
ECM[13]	Spatial And Temporal	Cache Line Transfers	Balancing Prefetching Aggressiveness
POM-FGIC[66]	Block-wise Communication	Thread-specific Data	Managing System Complexity
MOEACO[12]	Evolutionary Algorithms	Cache Configuration	Conflicting Objectives
MLOP[25]	Reduce Latency	Data Cache Management	Balancing Prefetching Aggressiveness
CCCS[27]	Coded Prefetching	Coded Subfiles	Prefetching Aggressiveness
XMem[26]	Cross-layer Abstractions	Efficient Memory Optimization	Balancing Prefetching Aggressiveness

Table 3: Comparison of various data prefetching methods, detailing their optimization techniques, memory management strategies, and associated performance challenges. The table highlights the diversity in approaches, from perceptron-based filtering to reinforcement learning, and underscores the ongoing challenges in balancing prefetching aggressiveness and accuracy in different computing environments.

Data prefetching techniques optimize computing performance by reducing latency and enhancing energy efficiency. Techniques like the Perceptron Prefetch Filter (PPF) enable aggressive prefetching with high accuracy, improving performance without resource wastage [14]. The HFetch framework optimizes read operations in multi-application workflows, reducing cache pollution and enhancing data access [58]. The Explicit Caching Hybrid (EHYB) method minimizes data movement, improving Sparse Matrix-Vector Multiplication (SpMV) operations [18].

Reinforcement learning-based approaches, such as ACOCA, adaptively manage context caching, optimizing performance through innovative techniques [11]. Neural prefetching methods manage larger working sets and complex access patterns, increasing prefetching strategy efficacy [15]. Information-theoretic caching methods efficiently manage cache memory and bandwidth, addressing existing technique limitations [9].

The Triage system offers reduced complexity and energy consumption in bandwidth-constrained environments due to on-chip metadata storage [20]. Efficient memory management techniques, like those in Kona, reduce memory waste and increase throughput by decoupling memory access tracking from virtual memory page size [52]. The Seer-MCache system improves cache hit rates and reduces latency for IoT queries, although it may increase resource consumption during heavy query loads [22]. The XKV method dynamically adjusts cache sizes, leading to significant memory savings and improved inference performance [23]. The proposed ISA extension approach reduces energy consumption by enabling smaller cache sizes without performance penalties [4].

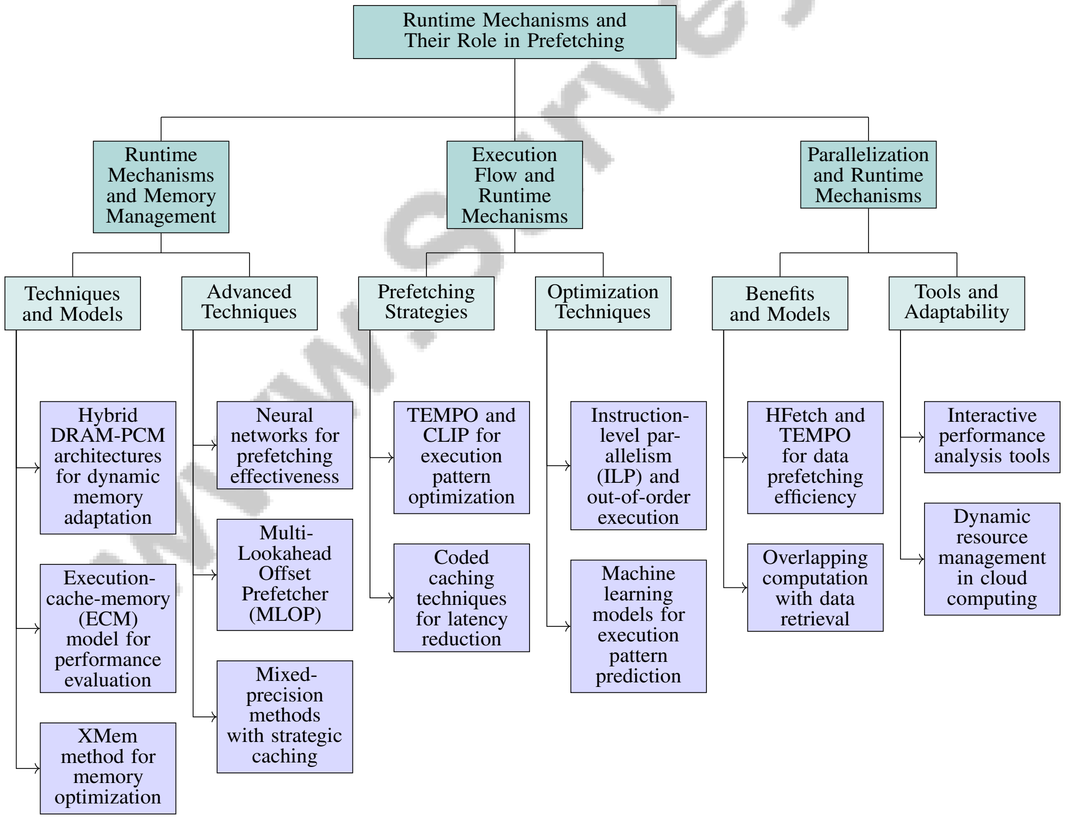
Despite these benefits, data prefetching techniques face challenges in balancing prefetching aggressiveness and accuracy. Excessive prefetching can lead to resource wastage and increased memory bandwidth consumption, while insufficient prefetching may fail to reduce latency adequately. The Execution-Cache-Memory (ECM) model offers insights into performance bottlenecks, highlighting the need for refined frameworks to predict and optimize performance in stencil algorithms [13]. Challenges in optimizing data access patterns and reducing remote memory access overhead remain critical considerations [66].

Multi-objective optimization approaches have shown significant improvements in execution time and energy consumption, with an average improvement of 64.43

The method proposed in [27] demonstrates improved delivery rates and efficient cache capacity use through coded prefetching. However, narrow interfaces pose significant drawbacks in achieving effective prefetching optimizations, emphasizing the need for richer abstractions [26].

Data prefetching techniques enhance computing performance and energy efficiency by anticipating future memory accesses and preloading data into faster storage. However, challenges include increased system complexity, adapting to varying access patterns, and managing memory resources effectively to avoid negatively impacting concurrent applications [29, 30, 58]. Continued research and development are essential to overcoming these limitations and enhancing prefetching strategy effectiveness in diverse environments. Table 3 provides a comprehensive overview of contemporary data prefetching methods, illustrating their respective optimization techniques, memory management strategies, and performance challenges, thereby elucidating the trade-offs and complexities inherent in optimizing computing performance and energy efficiency.

In examining the intricacies of memory management and computational efficiency, it is essential to consider the various runtime mechanisms that underpin these processes. Figure 4 illustrates the hierarchical structure of these runtime mechanisms and their critical role in prefetching. This figure highlights key techniques and models involved in memory management, execution flow, and parallelization. By emphasizing the importance of advanced strategies and tools, it underscores the necessity of optimizing memory access to improve computational efficiency and enhance overall system performance. Such visual representation not only aids in understanding the complex interrelationships among these components but also serves to reinforce the theoretical frameworks discussed throughout this paper.



Runtime Mechanisms and Their Role in Prefetching
Runtime Mechanisms and Memory Management
Execution Flow and Runtime Mechanisms
Parallelization and Runtime Mechanisms
Techniques and Models
Advanced Techniques
Prefetching Strategies
Optimization Techniques
Benefits and Models
Tools and Adaptability
Hybrid DRAM-PCM architectures for dynamic memory adaptation
Execution-cache-memory (ECM) model for performance evaluation
XMem method for memory optimization
Neural networks for prefetching effectiveness
Multi-Lookahead Offset Prefetcher (MLOP)
Mixed-precision methods with strategic caching
TEMPO and CLIP for execution pattern optimization
Coded caching techniques for latency reduction
Instruction-level parallelism (ILP) and out-of-order execution
Machine learning models for execution pattern prediction
HFetch and TEMPO for data prefetching efficiency
Overlapping computation with data retrieval
Interactive performance analysis tools
Dynamic resource management in cloud computing

Figure 4: This figure illustrates the hierarchical structure of runtime mechanisms and their role in prefetching, highlighting key techniques and models in memory management, execution flow, and parallelization. It emphasizes the importance of advanced strategies and tools for optimizing memory access, improving computational efficiency, and enhancing system performance.

AI-generated, for reference only.

Feature	Static and Dynamic Prefetching Methods	Hardware-Based Prefetching Approaches	Software-Based Prefetching Approaches
Adaptation Type	Static And Dynamic	Hardware-specific	Software-driven
Target Environment	All Applications	Many-core Systems	Flexible Environments
Optimization Focus	Data Access Efficiency	Latency Reduction	Data Accuracy

Table 4: This table provides a comparative analysis of different data prefetching methods, including static and dynamic, hardware-based, and software-based approaches. It highlights their adaptation types, target environments, and optimization focuses, offering insights into their applicability and efficiency in various computing contexts. Such a comparison is critical for understanding the strengths and limitations of each approach in optimizing data access and computational performance.

4 Runtime Mechanisms and Their Role in Prefetching

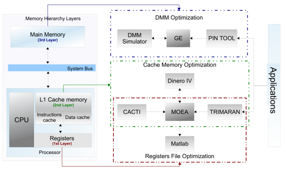
4.1 Runtime Mechanisms and Memory Management

Effective memory management is crucial for data prefetching, especially in complex computing environments where resource optimization is vital. Techniques in hybrid DRAM-PCM architectures dynamically adapt to memory access patterns, enhancing efficiency in graph-processing applications [18]. This adaptability is essential for managing competitive cache access, as illustrated by methods that allocate memory among threads to reduce cache misses [21]. Techniques optimizing response quality and resource usage through dynamic adjustments further demonstrate this adaptability [3].

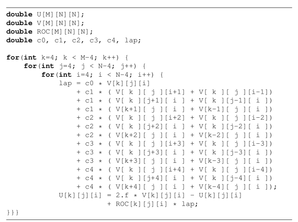
The execution-cache-memory (ECM) model provides a robust framework for performance evaluation by considering execution and data transfer times to identify bottlenecks, informing prefetching strategies [19]. This model is particularly beneficial in high-performance computing, where memory access optimization significantly influences system performance. The XMem method, which communicates higher-level program semantics for efficient memory optimization, underscores the importance of memory management in enhancing programmability, performance portability, and resource efficiency [26].

Neural networks improve prefetching effectiveness by modeling complex, non-linear data relationships, allowing for more accurate predictions of memory access patterns [15]. The Multi-Lookahead Offset Prefetcher (MLOP) maximizes miss coverage while ensuring timely prefetch requests [25]. In machine learning applications, mixed-precision methods combined with strategic caching reduce memory footprints while maintaining model accuracy, optimizing cache usage [4]. The Triage system exemplifies this by caching useful metadata on-chip, thus reducing off-chip traffic and energy consumption [20].

The scalability and efficiency of shared cache performance modeling highlight the importance of effective memory management in prefetching strategies, enabling accurate predictions without extensive simulations [14]. The Adaptive Memory Management Protocol (AMMP) excels by leveraging detailed analyses of memory reuse patterns and pipeline usage for accurate runtime predictions [28]. In cloud computing environments, understanding unique access patterns through adaptive prefetching and memory management strategies minimizes cache misses and enhances overall performance [2].



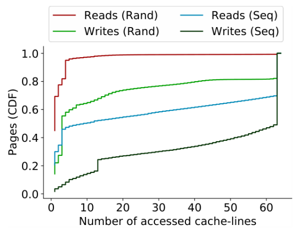
(a) Memory and Register Optimization Framework[70]



```
double U[M][N][N];
double V[M][N][N];
double ROC[M][N][N];
double c0, c1, c2, c3, c4, lap;

for(int k=4; k < M-4; k++) {
    for(int j=4; j < N-4; j++) {
        for(int i=4; i < N-4; i++) {
            lap = c0 * V[k][j][i]
                + c1 * ( V[ k ][ j ][i+1] + V[ k ][ j ][i-1])
                + c1 * ( V[ k ][j+1][ i ] + V[ k ][j-1][ i ])
                + c1 * ( V[k+1][ j ][ i ] + V[k-1][ j ][ i ])
                + c2 * ( V[ k ][ j ][i+2] + V[ k ][ j ][i-2])
                + c2 * ( V[ k ][j+2][ i ] + V[ k ][j-2][ i ])
                + c2 * ( V[k+2][ j ][ i ] + V[k-2][ j ][ i ])
                + c3 * ( V[ k ][ j ][i+3] + V[ k ][ j ][i-3])
                + c3 * ( V[ k ][j+3][ i ] + V[ k ][j-3][ i ])
                + c3 * ( V[k+3][ j ][ i ] + V[k-3][ j ][ i ])
                + c4 * ( V[ k ][ j ][i+4] + V[ k ][ j ][i-4])
                + c4 * ( V[ k ][j+4][ i ] + V[ k ][j-4][ i ])
                + c4 * ( V[k+4][ j ][ i ] + V[k-4][ j ][ i ]);
            U[k][j][i] = 2.f * V[k][j][i] - U[k][j][i]
                       + ROC[k][j][i] * lap;
}}}
```

(b) A C++ code snippet for calculating the Laplacian matrix of a graph[71]



(c) Performance of Sequential and Random Access in Cache Lines[52]

Figure 5: Examples of Runtime Mechanisms and Memory Management

As shown in Figure 5, the exploration of runtime mechanisms reveals their critical role in prefetching and memory management. The "Memory and Register Optimization Framework" flowchart delineates

AI-generated, for reference only.

stages and tools integral to optimizing memory usage, emphasizing the interplay between memory hierarchy layers and the system bus. The "C++ code snippet for calculating the Laplacian matrix" illustrates how runtime mechanisms can be leveraged for efficient computation by iterating over graph vertices and edges, highlighting the importance of arithmetic operations and code annotations. Lastly, the "Performance of Sequential and Random Access in Cache Lines" graph compares access patterns, underscoring the impact of access strategies on memory performance and the role of runtime mechanisms in optimizing these strategies.

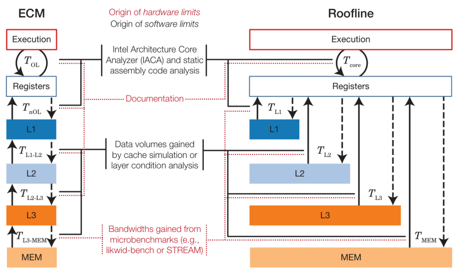
4.2 Execution Flow and Runtime Mechanisms

Execution flow significantly influences runtime mechanisms, particularly in data prefetching, as it dictates the timing and accuracy of memory access predictions. Effective prefetching strategies implemented in systems like TEMPO and CLIP utilize execution patterns to optimize memory access, reducing latency and enhancing performance. TEMPO improves data access efficiency by prefetching data pointed to by page tables, while CLIP enhances prefetch accuracy in bandwidth-constrained environments, illustrating the critical relationship between execution flow and memory optimization techniques [29, 32, 31, 45].

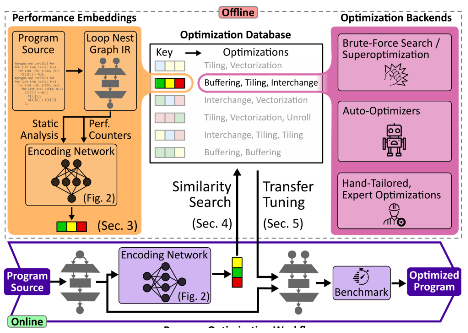
Aligning instruction execution with data retrieval is pivotal for optimizing prefetching, directly impacting data availability and system performance. A primary challenge is ensuring data is available precisely when needed, necessitating a dynamic understanding of execution patterns and accurate predictions of future data demands. Coded caching techniques enhance delivery latency by introducing redundancy through coded storage, particularly effective in random user-server connections [72].

Execution flow optimization also involves strategic instruction scheduling to minimize stalls and maximize resource utilization. Techniques such as instruction-level parallelism (ILP) and out-of-order execution improve throughput by allowing concurrent processing of multiple instructions and non-sequential execution. These optimizations reduce idle time and address performance bottlenecks, especially in multi-core architectures. Advanced data prefetching strategies, including static and dynamic prefetchers, enhance performance by anticipating future memory requests, ensuring timely data availability [20, 56, 35, 25, 33].

Advanced runtime mechanisms utilize machine learning models to predict execution patterns and adjust prefetching strategies accordingly. These models leverage historical data access patterns and execution traces to identify optimal prefetching points, significantly improving the accuracy and timeliness of data retrieval processes, thereby addressing latency challenges faced by high-performance processors in server workloads [29, 21, 30, 34].


ECM
Origin of hardware limits
Origin of software limits
Roofline
Execution
Registers
L1
L2
L3
MEM
T_{OL}
T_{nOL}
$T_{L1\text{-}L2}$
$T_{L2\text{-}L3}$
$T_{L3\text{-}MEM}$
T_{core}
T_{L1}
T_{L2}
T_{L3}
T_{MEM}
Intel Architecture Core Analyzer (IACA) and static assembly code analysis
Documentation
Data volumes gained by cache simulation or layer condition analysis
Bandwidths gained from microbenchmarks (e.g., likwid-bench or STREAM)

(a) Understanding the Execution and Memory Latency in Modern Hardware and Software[71]



(b) Program Optimization Workflow[73]

Figure 6: Examples of Execution Flow and Runtime Mechanisms

As shown in Figure 6, two illustrative examples highlight the role of execution flow in runtime mechanisms. The first, "Understanding the Execution and Memory Latency," provides a diagram elucidating execution and memory access stages in modern hardware and software, emphasizing tools like the Intel Architecture Core Analyzer (IACA) and methods for evaluating execution flow and memory latency. The second example, "Program Optimization Workflow," visually represents a systematic approach to optimizing program performance, detailing offline and online stages and opti-

AI-generated, for reference only.

mization techniques. Together, these examples underscore the intricate interplay between execution flow, runtime mechanisms, and strategies for enhancing computational efficiency and performance through effective prefetching and optimization techniques.

4.3 Parallelization and Runtime Mechanisms

Parallelization enhances data prefetching efficiency by distributing tasks across multiple processing units, improving throughput and reducing execution time in data-intensive computing. Hierarchical data prefetching approaches, such as HFetch, utilize a server-push model to optimize data delivery, achieving performance gains exceeding 1000

A key benefit of parallelization in prefetching is the ability to overlap computation with data retrieval, minimizing idle time and maximizing resource utilization. Concurrent execution of multiple threads enables effective prefetching alongside ongoing computations, reducing memory access latency. This model is particularly advantageous for applications with high data parallelism, allowing task decomposition into smaller, independent units for simultaneous execution. Such approaches enhance processing efficiency and mitigate performance penalties associated with long latency cache misses, as demonstrated by various techniques optimizing memory access patterns [30, 25, 31, 53, 21].

Interactive performance analysis tools, as highlighted by Schaad et al., provide immediate feedback on performance characteristics, facilitating quicker optimization decisions and reducing cognitive load [19]. These tools help identify bottlenecks and inefficiencies in parallel execution, enabling developers to fine-tune prefetching strategies and runtime mechanisms for optimal performance.

Moreover, parallelization enhances the adaptability of prefetching mechanisms by allowing real-time adjustments based on workload characteristics. Advanced scheduling algorithms allocate resources to prefetching tasks based on priority and demand, ensuring critical data availability. Dynamic resource management is crucial in cloud computing environments with fluctuating workloads, involving mechanisms like Scheduling-Aware Data Prefetching (SADP) and PagedAttention to efficiently cache frequently accessed data in low-latency storage while evicting less-used datasets to higher-latency storage. This strategy improves data retrieval efficiency and minimizes memory fragmentation, enhancing throughput for diverse applications without compromising overall system performance [30, 31, 53].

5 Hardware Architecture and Prefetching

5.1 Impact of CPU Architectures on Prefetching

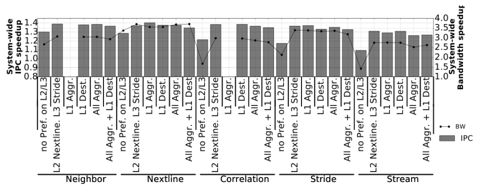
CPU architectures significantly influence prefetching strategies, affecting system performance by shaping data retrieval mechanisms and handling diverse memory access patterns. Evaluations of Kerncraft on Intel Sandy Bridge and Haswell architectures highlight the necessity of CPU-specific optimizations in prefetching, requiring tools that adapt to unique hardware characteristics [59]. This adaptability addresses challenges posed by multicore processors, where cache access contention can lead to competitive cache misses, necessitating tailored strategies to maintain performance [6].

The complexity of predicting optimal prefetch distances and timings is compounded by the variability in memory access patterns across CPU architectures. The RPG 2 framework dynamically adjusts prefetching strategies based on program input and hardware architecture, enhancing accuracy and efficiency [45]. This dynamic approach is crucial in heterogeneous workloads, where static techniques may be insufficient.

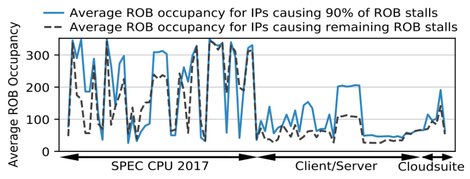
CPU architecture's influence is further demonstrated through cache configuration evaluations on an Intel Core i7-3770, showing how design impacts prefetching strategies and energy consumption [12]. The Domino prefetching method, assessed on a quad-core processor, illustrates the impact of CPU design on managing cache misses and optimizing data access times [21].

The limitations of general-purpose CPUs and GPUs in adapting to diverse application demands underscore the need for architecture-specific strategies. Liu et al. suggest that customized architectures could better cater to unique application requirements [55]. This is particularly relevant in cloud computing, where distinct access patterns necessitate prefetching strategies beyond traditional techniques [2].

Moreover, the interaction between CPU design and cross-layer abstractions is critical in shaping prefetching strategies. Vijaykumar et al. emphasize the importance of integrating architectural considerations into methodologies to enhance programmability, portability, and efficiency [26].



(a) Performance of network aggregation strategies on a single CPU with two interfaces.[35]



(b) Average ROB occupancy for IPs causing 90% of ROB stalls[74]

Figure 7: Impact of CPU Architectures on Prefetching

As shown in Figure 7, understanding CPU architecture's influence on prefetching is vital for optimizing system performance. The figures illustrate key aspects of this relationship by analyzing network aggregation strategies and their effects on efficiency. The first figure evaluates performance in a system with a single CPU and dual interfaces, highlighting network aggregation's significance at various cache levels, such as L2 and L3, on prefetching efficiency. The second figure examines average ROB stalls, comparing occupancy for IPs responsible for most stalls, providing insights into how specific IPs affect prefetching performance. Together, these figures offer a comprehensive perspective on how CPU architecture choices impact prefetching strategies, guiding enhancements in computational efficiency and resource management [35, 74].

5.2 Memory Hierarchies and Prefetching Strategies

Memory hierarchies critically shape prefetching strategies, dictating data access organization and efficiency. Cache memory configuration, a key hierarchy component, can be optimized based on application profiling to enhance prefetching effectiveness [70]. This involves tailoring cache configurations to specific application needs, improving retrieval times and reducing latency.

Application-Specific Instruction-set Processor (ASIP) simulators provide detailed cache simulation results tailored to applications [75]. These simulators offer insights into cache behavior, informing prefetching strategies that align with unique application requirements, ensuring efficient resource utilization.

A systematic simulation methodology proposed by Bueno et al. enhances cache performance evaluation accuracy compared to conventional techniques [36]. By minimizing changes in cache policy order by over 30%, this methodology establishes a reliable basis for developing responsive prefetching strategies.

In cloud computing, understanding cache configurations' impact on performance is crucial for optimizing prefetching strategies [76]. Selected metrics provide insights into how cache settings influence data access patterns and efficiency, essential for designing approaches that manage data movement in resource-constrained, dynamic environments.

Yang et al. propose an architecture addressing limited high-bandwidth memory capacity in GPUs, significantly reducing training costs for large-scale embeddings [10]. By optimizing cache usage through mixed-precision techniques, this approach enhances prefetching and reduces data-intensive operation overhead.

Analyzing cache replacement policies like FIFO, PLRU, and NMRU presents intrinsic challenges due to distinct operational characteristics [49]. Selecting appropriate prefetching strategies that align with specific policies is crucial for optimal cache performance and data access efficiency.

Architecture-independent reuse distance analysis allows accurate shared cache performance predictions without extensive trace collection [54]. This capability is crucial for developing adaptable prefetching strategies that efficiently manage data across diverse environments.

New cache utilization strategies analyzed by Majumdar show significant improvements over traditional methods [56]. Enhancing cache utilization contributes to effective prefetching approaches, reducing retrieval times and improving system performance.

The relationship between memory hierarchies and prefetching strategies is highlighted by cache size's impact on dynamic energy savings [4]. Understanding this relationship enables developers to design strategies optimizing energy consumption while maintaining high performance, ensuring efficient and sustainable systems.

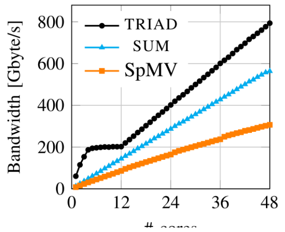
5.3 Impact of GPGPU and Multi-GPU Architectures

The influence of General-Purpose Graphics Processing Unit (GPGPU) and multi-GPU architectures on data prefetching is profound, presenting unique challenges and opportunities for optimizing data access and computational efficiency. A primary obstacle in adapting strategies for GPGPU architectures is the inadequacy of existing CPU cache optimization techniques, which are not sufficiently tailored or tested for GPGPU systems, resulting in underutilization [61].

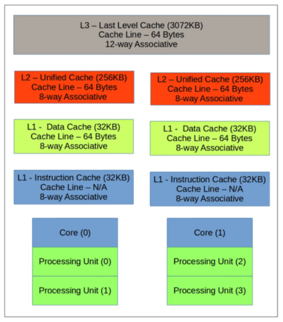
In multi-GPU systems, performance bottlenecks often arise from limited inter-GPU interconnect bandwidth and complexities in managing remote memory accesses. Sophisticated prefetching techniques are needed to efficiently manage data transfers between GPUs, ensuring effective resource utilization and timely data availability [77]. Prefetching across multiple GPUs can enhance performance by reducing latency and minimizing idle time, improving throughput.

The complexity of GPU architectures and programming models complicates developing effective prefetching strategies. These systems require advanced performance tools providing detailed insights into bottlenecks, guiding optimization of data prefetching techniques. Such tools are essential for leveraging GPU compute power and optimizing data access patterns for GPU workloads [78].

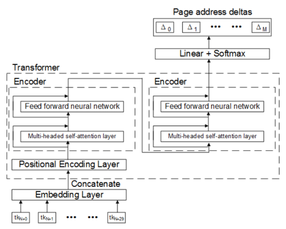
The influence of GPGPUs and multi-GPU architectures on prefetching underscores the need for continuous innovation and refinement in methodologies, particularly within Unified Virtual Memory (UVM) systems. Recent advancements, like deep learning-based approaches, show significant improvements in prediction accuracy and memory access efficiency, indicating that adaptive strategies encompassing local and global patterns are essential for optimizing performance in complex environments [29, 32, 14, 44]. By addressing specific challenges posed by these architectures, researchers can develop strategies enhancing computational efficiency, reducing latency, and maximizing GPU-based systems' performance potential.



(a) Performance Comparison of TRIAD, SUM, and SpMV Algorithms on GPU[40]



(b) Block diagram of computer architecture focusing on cache hierarchy and memory organization.[61]



(c) Transformer-based Page Address Prediction Model[44]

Figure 8: Impact of GPGPU and Multi-GPU Architectures

As shown in Figure 8, the examples illustrate the intricate interplay between hardware architecture and prefetching, examining the impact of GPGPU and multi-GPU architectures. The first figure provides a performance comparison of TRIAD, SUM, and SpMV algorithms on a GPU, emphasizing core count's role in bandwidth efficiency. The second figure focuses on computer architecture's structural elements, specifically cache hierarchy and memory organization, highlighting cache design's role in data flow and access speed optimization. The third figure introduces a transformer-based page address prediction model, using advanced machine learning techniques to predict memory access patterns, exemplifying cutting-edge approaches to enhance memory management efficiency in GPGPU and multi-GPU systems [40, 61, 44].

5.4 Hardware-Software Co-Design for Prefetching

The hardware-software co-design paradigm is crucial for optimizing data prefetching by integrating hardware architectures and software algorithms' strengths. This synergy enhances data access efficiency and system performance, particularly in high-performance environments where memory latency is a bottleneck. Techniques like load criticality predictors and translation-triggered prefetching address challenges of constrained DRAM bandwidth and complex patterns, improving responsiveness in server workloads and data-intensive applications [20, 29, 32, 31]. This approach integrates hardware capabilities with software intelligence to create adaptive prefetching mechanisms.

A key advantage of co-design is tailoring strategies to specific application requirements and hardware configurations. Software algorithms analyze patterns, allowing systems to dynamically adjust hardware parameters for optimized retrieval. This adaptability is exemplified by ReSemble, enhancing performance through real-time learning and adaptability to diverse patterns [17].

The integration of hardware and software is illustrated by the Caching Remote Data in Video Memory (CARVE) method, transforming GPU memory into a hybrid structure by storing recently accessed remote data in a dedicated region. This reduces NUMA bottlenecks and improves access times, highlighting hardware-software co-design's potential in optimizing prefetching for GPU architectures [77].

Effectiveness is supported by methodologies capturing higher Last-Level Cache (LLC) activity, leading to accurate performance predictions and improved comparisons between cache policies [36]. Aligning hardware capabilities with software insights achieves precise data management and improved cache utilization.

Hardware-software co-design provides a robust framework for enhancing prefetching strategies, enabling efficient memory access management by integrating advanced techniques like load criticality prediction and filtering mechanisms, improving coverage and accuracy, thus mitigating bottlenecks associated with latency [29, 32, 14]. By merging hardware efficiencies with software intelligence, this approach enhances data access efficiency, reduces latency, and maximizes performance, addressing contemporary applications' evolving demands.

6 Performance Optimization Strategies

Modern computing systems face significant challenges in performance optimization, necessitating diverse strategies to enhance execution efficiency and resource use. This section explores key methodologies, starting with data prefetching techniques, which anticipate data needs and reduce memory latency, thereby cutting execution time and boosting computational efficiency. The following subsection examines data prefetching's crucial role in optimizing execution across various computing environments.

6.1 Data Prefetching and Execution Time Reduction

Data prefetching enhances computational efficiency by anticipating data needs and minimizing memory latency across various environments. The Multi-Lookahead Offset Prefetcher (MLOP) exemplifies advanced prefetching, achieving a 30% improvement over baseline systems [25]. In content delivery networks, caching schemes by Gómez-Vilardebó et al. and Amiri et al. improve delivery rates and execution time [27, 24]. Truffle's integration of data transfer with the cold start process enhances data-intensive application performance [3]. In scientific computing, the Adaptive Memory Management Protocol (AMMP) significantly improves execution times [28]. XMem improves execution times by 8.5%, enhancing programmability and resource efficiency [26]. The GPU Performance Advisor (GPA) demonstrates potential performance gains through targeted optimizations [78]. In web applications, Google's Core Web Vitals, edge computing, and local-first software enhance responsiveness and execution times [5]. Recent advancements in scheduling-aware and hierarchical data prefetching demonstrate substantial performance gains in cloud computing and scientific workflows [29, 30, 58, 14].

6.2 Energy Efficiency and Resource Management

Data prefetching is pivotal for enhancing energy efficiency and resource management in modern computing systems. Prefetching strategies reduce energy consumption by predicting data access patterns and minimizing unnecessary memory operations. Techniques like Translation-Enabled Memory Prefetching (TEMPO) and Perceptron-based Prefetch Filtering (PPF) can cut energy use by up to 14

6.3 Cache Optimization Techniques

Optimizing cache usage is vital for enhancing system performance, especially in multi-core and application-specific environments. Application-Specific Instruction-set Processor (ASIP) simulators, like SimpleScalar and VEX, provide insights into cache designs, enabling tailored performance optimization [75]. These simulators aid cache performance analysis across various Instruction Set Architectures (ISAs) [51]. The Bingo prefetcher exemplifies advanced cache optimization, achieving over 63

6.4 Hardware and Software Synergy

The synergy between hardware and software is crucial for achieving significant performance gains by integrating hardware capabilities with software intelligence. Kerncraft automates performance modeling through hardware knowledge and source code analysis, providing developers with precise performance predictions [59, 71]. In multimedia applications, tailoring hardware configurations through multi-objective optimization enhances energy efficiency and execution time [12]. The Seer-MCache system exemplifies hardware-software synergy with its modular design, optimizing performance based on varying workloads [22]. Performance embeddings facilitate knowledge transfer of optimizations, enabling identification of performance patterns that inform strategies [73]. A proposed metric for characterizing memory access in parallel architectures provides insights into optimization strategies [41]. The XMem interface enhances hardware-software synergy by allowing for intelligent resource management [26]. In database interactions, benchmarks simulate realistic access patterns, informing optimization strategies [42]. The synergy between hardware and software is critical for enhancing resource utilization, minimizing latency, and improving overall performance, especially in high-performance environments like data centers where optimized data prefetching strategies address memory access bottlenecks [56, 29, 26, 65].

6.5 Scalability and Adaptability in Optimization

Scalability and adaptability are essential for maximizing computing efficiency, especially as systems evolve towards exascale computing. Developing novel optimization techniques for managing data exchanges and achieving load balancing is crucial [16]. Future research should focus on optimizing MLOP's hardware implementation for dynamic environments [25]. In multicore environments, optimizing virtual cache allocation strategies can mitigate competitive cache misses and improve performance [6]. The Khameleon framework could benefit from enhanced prediction models and scheduling policies for improved scalability [8]. The ECM method's inherent scalability suggests further exploration of software techniques and hardware configurations to enhance energy efficiency [39]. Dynamic caching strategies, as suggested by information-theoretic approaches, can further improve adaptability [9]. Performance modeling frameworks should accommodate irregular patterns and parallel programming environments to enhance scalability [66]. Refining models to incorporate latency penalties and applying them to complex stencil optimizations can contribute to scalable strategies [13]. Future research directions include exploring optimizations in decentralized caching systems and integrating performance optimizations into existing frameworks for web applications [5]. Optimizing centralized coded caching schemes for scalability and adaptation to varying content popularity distributions is another area for development [27]. Additionally, future developments will focus on integrating comprehensive compiler information into tools like the GPU Performance Advisor for improved speedup estimates and exploring additional optimization patterns [78]. This focus on scalability and adaptability is critical for ensuring optimization techniques remain effective, facilitating efficient resource use across diverse technological landscapes.

7 Cache Memory and Its Impact on Prefetching

Cache memory is fundamental in computing, significantly affecting the efficiency of data prefetching mechanisms. Understanding the interplay between cache characteristics and prefetching strategies is crucial for optimizing system performance. This section delves into critical aspects of cache memory, emphasizing cache size and architecture, which are pivotal in shaping prefetching effectiveness.

7.1 Role of Cache Size and Architecture

Cache size and architecture are integral to designing effective prefetching strategies, directly impacting data retrieval efficiency and overall system performance. Data Direct I/O (DDIO) exemplifies this by reducing data access latency, allowing I/O devices direct access to the Last Level Cache (LLC), enhancing throughput [68]. Techniques that optimize cache usage during data loading minimize competitive cache misses, thus improving performance [6]. Additionally, custom instructions that reduce cache accesses underscore the importance of cache size and architecture in achieving energy savings without compromising performance [4].

7.2 Cache Speed and Prefetching Efficiency

Cache speed is a critical determinant of data prefetching effectiveness, influencing how swiftly data can be accessed by the CPU. High-speed caches reduce latency, ensuring timely data access and enhancing efficiency. Advanced prefetching techniques, such as Perceptron-based Prefetch Filtering and on-chip metadata storage, optimize memory access and mitigate cache miss-related bottlenecks, improving performance in memory-intensive workloads [20, 29, 79, 31, 14]. High-speed caches enable aggressive prefetching, crucial in high-throughput settings where delays can cause bottlenecks. Techniques like Scheduling-Aware Data Prefetching (SADP) and hierarchical prefetching manage data across multi-tiered systems, prioritizing low-latency access to frequently used datasets [29, 58, 30, 31, 14]. Sophisticated cache management, including optimized replacement policies and predictive algorithms, significantly enhances prefetching efficiency [34, 31, 14].

7.3 Innovative Cache Management Techniques

Innovative cache management techniques are crucial for improving data prefetching efficiency by enabling systems to anticipate and fulfill data requests promptly. Advanced replacement policies, such as the Adaptive Weight Ranking Policy (AWRP), optimize cache storage by ranking pages based on recency and frequency, thereby improving cache hit ratios [37]. Machine learning models, like neural prefetching, predict future memory accesses based on historical patterns, reducing cache misses and improving retrieval times [15]. Dynamic cache management techniques, such as AdaptSize, adjust caching parameters in real-time based on workload characteristics, maximizing Object Hit Ratio (OHR) [57]. Hybrid cache architectures, which combine traditional elements with computational capabilities, enhance prefetching efficiency by reducing data movement overhead [7]. These techniques are pivotal in improving data retrieval, reducing latency, and enhancing performance in high-performance server applications [29, 14].

7.4 Impact of Cache Hit and Miss Rates

Cache hit and miss rates significantly impact prefetching performance and overall system efficiency. High cache hit rates indicate successful data retrieval from cache memory, reducing access times and enhancing performance. Conversely, low hit rates increase latency as the system fetches data from slower main memory. Effective prefetching techniques, such as translation-enabled memory prefetching (TEMPO) and perceptron-based filtering, aim to improve hit rates by anticipating future data needs based on historical access patterns, optimizing memory access efficiency [56, 34, 31, 14]. High cache miss rates pose challenges, leading to increased latency and reduced performance. Prefetching strategies mitigate these issues by anticipating future data needs, although accuracy and timeliness are crucial to avoid cache pollution and increased miss rates. Machine learning models have shown promise in reducing cache miss rates by predicting future data accesses [15]. Adaptive cache management strategies dynamically adjust caching parameters based on real-time workload characteristics, improving hit rates and reducing miss rates [57].

7.5 Systematic Evaluation of Cache Performance

Benchmark	Size	Domain	Task Format	Metric
ECM[80]	100	Energy-efficient Computing	Performance Prediction	Energy-Delay Product, Runtime
ECM[81]	500,000	Quantum Chromodynamics	Sparse Matrix-Vector Multiplication	Gflop/s, Memory Bandwidth
DPIOP[35]	9	High Performance Computing	Data Prefetching	Accuracy, Timeliness
CACHE-MEM[51]	1,000,000	Computer Architecture	Performance Evaluation	Access Time, Execution Time

Table 5: The table provides a comprehensive overview of representative benchmarks utilized in the evaluation of cache performance strategies. It details the benchmark name, dataset size, domain of application, task format, and performance metrics, highlighting the diversity and scope of benchmarks in energy-efficient computing, quantum chromodynamics, high-performance computing, and computer architecture.

Systematic evaluation of cache performance is vital for optimizing prefetching strategies. Table 5 presents a detailed enumeration of various benchmarks employed in the systematic evaluation of cache performance, illustrating the wide range of domains and metrics considered in optimizing prefetching strategies. Accurate evaluation methods provide insights that facilitate the design of advanced prefetching techniques, balancing cache hit and miss rates to minimize latency and enhance throughput. Innovations like Perceptron-Based Prefetch Filtering (PPF) improve prefetch coverage while maintaining accuracy, achieving performance gains in multi-core configurations. The Multi-Lookahead Offset Prefetcher (MLOP) enhances performance by intelligently selecting prefetch offsets, underscoring the importance of sophisticated strategies in mitigating cache miss penalties [79, 34, 25, 14]. Simulation methodologies that capture dynamic cache operations are valuable for evaluating performance. Bueno et al. propose a systematic simulation methodology that enhances the representativeness of cache performance evaluations, providing a reliable basis for assessing prefetching strategies [36]. Performance modeling tools, such as the Execution-Cache-Memory (ECM) model, optimize memory access by considering execution and data transfer times, particularly in high-performance computing scenarios [39]. Integrating machine learning models into performance evaluation enhances prefetching strategies, predicting future memory accesses to reduce cache misses and improve retrieval times [15]. Adaptive cache management techniques, like AdaptSize, continuously adjust caching parameters to optimize hit ratios and maintain high cache efficiency [57]. Comprehensive evaluation of cache performance is crucial for optimizing prefetching strategies, balancing coverage and accuracy to enhance system efficiency and reduce resource waste in modern processors. By leveraging advanced methodologies, performance modeling tools, machine learning models, and adaptive management techniques, modern computing systems can achieve significant improvements in data retrieval efficiency, reduce latency, and enhance overall performance, addressing the evolving demands of contemporary computing environments [20, 29, 34, 31, 14].

8 Conclusion

Data prefetching plays a pivotal role in modern computing by mitigating memory latency and enhancing computational efficiency. This survey underscores the importance of prefetching techniques, evaluating their capabilities and constraints. Models like the Execution-Cache-Memory (ECM) provide insights into performance and energy dynamics, highlighting areas for optimization. The adaptability of database management systems and clustering algorithms to dynamic access patterns, as illustrated by the dynamic object evaluation framework (DoEF), merits further exploration.

Promising research directions include exploring heterogeneous precision assignments for embeddings and examining cache policy impacts on performance. The evolution of prefetching techniques continues to improve metrics such as inference latency and power consumption. For instance, AdaptSize optimizes Object Hit Ratio (OHR) by managing diverse request patterns, while MERAM technology offers competitive performance and energy efficiency, making it suitable for L2 cache applications.

A thorough understanding of memory access patterns is vital for performance enhancement, suggesting further architectural adaptations based on identified Gene Patterns. Research could also focus on reducing preprocessing times and optimizing various matrix structures in prefetching contexts. The significance of XMem in boosting performance and software optimization portability across hardware architectures indicates potential for improved programmability and resource efficiency.

In serverless computing, the efficiency of techniques like Truffle, which achieves substantial latency reductions, highlights the potential for extending adaptive memory management protocols to integrate higher-level parallelism models and apply them across a wider array of scientific codes and architectures.

References

[1] Mengzhao Zhang, Qian Tang, Jeong-Geun Kim, Bernd Burgstaller, and Shin-Dug Kim. Adaptive regression prefetching algorithm by using big data application characteristics. *Applied Sciences*, 13(7):4436, 2023.

[2] Jiajun Wang, Reena Panda, and Lizy Kurian John. Prefetching for cloud workloads: An analysis based on address patterns. In *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 163–172. IEEE, 2017.

[3] Cynthia Marcelino and Stefan Nastic. Truffle: Efficient data passing for data-intensive serverless workflows in the edge-cloud continuum, 2024.

[4] Noushin Behboudi, Mehdi Kamal, and Ali Afzali-Kusha. On the impact of isa extension on energy consumption of i-cache in extensible processors, 2024.

[5] Juho Vepsäläinen, Arto Hellas, and Petri Vuorimaa. Overview of web application performance optimization techniques, 2024.

[6] Milcho Prisagjanec and Pece Mitrevski. Reducing competitive cache misses in modern processor architectures, 2017.

[7] Shaizeen Aga, Supreet Jeloka, Arun Subramaniyan, Satish Narayanasamy, David Blaauw, and Reetuparna Das. Compute caches. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 481–492. IEEE, 2017.

[8] Haneen Mohammed. Continuous prefetch for interactive data applications. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 2841–2843, 2020.

[9] Sung Hoon Lim, Chien-Yi Wang, and Michael Gastpar. Information theoretic caching: The multi-user case, 2016.

[10] Jie Amy Yang, Jianyu Huang, Jongsoo Park, Ping Tak Peter Tang, and Andrew Tulloch. Mixed-precision embedding using a cache, 2020.

[11] Shakthi Weerasinghe, Arkady Zaslavsky, Seng W. Loke, Amin Abken, and Alireza Hassani. Reinforcement learning based approaches to adaptive context caching in distributed context management systems, 2023.

[12] Josefa Díaz Álvarez, José L. Risco-Martín, and J. Manuel Colmenar. Multi-objective optimization of energy consumption and execution time in a single level cache memory for embedded systems, 2023.

[13] Holger Stengel, Jan Treibig, Georg Hager, and Gerhard Wellein. Quantifying performance bottlenecks of stencil computations using the execution-cache-memory model, 2015.

[14] Eshan Bhatia, Gino Chacon, Seth Pugsley, Elvira Teran, Paul V Gratz, and Daniel A Jiménez. Perceptron-based prefetch filtering. In *Proceedings of the 46th International Symposium on Computer Architecture*, pages 1–13, 2019.

[15] Milad Hashemi, Kevin Swersky, Jamie A. Smith, Grant Ayers, Heiner Litz, Jichuan Chang, Christos Kozyrakis, and Parthasarathy Ranganathan. Learning memory access patterns, 2018.

[16] Claude Tadonki. High performance optimization at the door of the exascale, 2021.

[17] Pengmiao Zhang, Rajgopal Kannan, Ajitesh Srivastava, Anant V Nori, and Viktor K Prasanna. Resemble: reinforced ensemble framework for data prefetching. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–14. IEEE, 2022.

[18] Chong Chen. Explicit caching hyb: a new high-performance spmv framework on gpgpu, 2022.

[19] Philipp Schaad, Tal Ben-Nun, and Torsten Hoefler. Boosting performance optimization with interactive data movement visualization, 2022.

[20] Hao Wu, Krishnendra Nathella, Joseph Pusdesris, Dam Sunwoo, Akanksha Jain, and Calvin Lin. Temporal prefetching without the off-chip metadata. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 996–1008, 2019.

[21] Mohammad Bakhshalipour, Pejman Lotfi-Kamran, and Hamid Sarbazi-Azad. Domino temporal data prefetcher. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 131–142. IEEE, 2018.

[22] Dingding Li, Mianxiong Dong, Yanting Yuan, Jiaxin Chen, Kaoru Ota, and Yong Tang. Seer-mcache: A prefetchable memory object caching system for iot real-time data processing. *IEEE Internet of Things Journal*, 5(5):3648–3660, 2018.

[23] Weizhuo Li, Zhigang Wang, Yu Gu, and Ge Yu. Xkv: Personalized kv cache memory reduction for long-context llm inference, 2024.

[24] Mohammad Mohammadi Amiri and Deniz Gunduz. Fundamental limits of coded caching: Improved delivery rate-cache capacity trade-off, 2016.

[25] Mehran Shakerinava, Mohammad Bakhshalipour, Pejman Lotfi-Kamran, and Hamid Sarbazi-Azad. Multi-lookahead offset prefetching. *The Third Data Prefetching Championship*, (2019), 2019.

[26] Nandita Vijaykumar. Enhancing programmability, portability, and performance with rich cross-layer abstractions, 2019.

[27] Jesús Gómez-Vilardebó. Fundamental limits of caching: Improved bounds with coded prefetching, 2017.

[28] Gopinath Chennupati, Nandakishore Santhi, Phill Romero, and Stephan Eidenbenz. Machine learning enabled scalable performance prediction of scientific codes, 2020.

[29] Mohammad Bakhshalipour, Mehran Shakerinava, Fatemeh Golshan, Ali Ansari, Pejman Lotfi-Karman, and Hamid Sarbazi-Azad. A survey on recent hardware data prefetching approaches with an emphasis on servers. *arXiv preprint arXiv:2009.00715*, 2020.

[30] Chien-Hung Chen, Ting-Yuan Hsia, Yennun Huang, and Sy-Yen Kuo. Scheduling-aware data prefetching for data processing services in cloud. In *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*, pages 835–842. IEEE, 2017.

[31] Abhishek Bhattacharjee. Translation-triggered prefetching. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 63–76, 2017.

[32] Biswabandan Panda. Clip: Load criticality based data prefetching for bandwidth-constrained many-core systems. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 714–727, 2023.

[33] Wim Heirman, Kristof Du Bois, Yves Vandriessche, Stijn Eyerman, and Ibrahim Hur. Near-side prefetch throttling: Adaptive prefetching for high-performance many-core processors. In *Proceedings of the 27th international conference on parallel architectures and compilation techniques*, pages 1–11, 2018.

[34] Haoyuan Wang and Zhiwei Luo. Data cache prefetching with perceptron learning. *arXiv preprint arXiv:1712.00905*, 2017.

[35] Cristobal Ortega, Victor Garcia, Miquel Moreto, Marc Casas, and Roxana Rusitoru. Data prefetching on in-order processors. In *2018 International Conference on High Performance Computing & Simulation (HPCS)*, pages 322–329. IEEE, 2018.

[36] Nicolas Bueno, Fernando Castro, Luis Pinuel, Jose Ignacio Gomez-Perez, and Francky Catthoor. Improving the representativeness of simulation intervals for the cache memory system, 2024.

[37] Debabala Swain, Bijay Paikaray, and Debabrata Swain. Awrp: Adaptive weight ranking policy for improving cache performance, 2011.

[38] Sanket Tavarageri, Gagandeep Goyal, Sasikanth Avancha, Bharat Kaul, and Ramakrishna Upadrasta. Ai powered compiler techniques for dl code optimization, 2021.

[39] Johannes Hofmann and Dietmar Fey. An ecm-based energy-efficiency optimization approach for bandwidth-limited streaming kernels on recent intel xeon processors, 2016.

[40] Christie L. Alappat, Jan Laukemann, Thomas Gruber, Georg Hager, Gerhard Wellein, Nils Meyer, and Tilo Wettig. Performance modeling of streaming kernels and sparse matrix-vector multiplication on a64fx, 2020.

[41] Aditya Chilukuri, Josh Milthorpe, and Beau Johnston. Characterizing optimizations to memory access patterns using architecture-independent program features, 2020.

[42] Zhen He and Jérôme Darmont. Evaluating the dynamic behavior of database applications, 2017.

[43] Arsalan Shahid, Muhammad Tayyab, Muhammad Yasir Qadri, Nadia N. Qadri, and Jameel Ahmed. Analytical models of energy and throughput for caches in mpsocs, 2019.

[44] Xinjian Long, Xiangyang Gong, Bo Zhang, and Huiyang Zhou. Deep learning based data prefetching in cpu-gpu unified virtual memory. *Journal of Parallel and Distributed Computing*, 174:19–31, 2023.

[45] Yuxuan Zhang, Nathan Sobotka, Soyoon Park, Saba Jamilan, Tanvir Ahmed Khan, Baris Kasikci, Gilles A Pokam, Heiner Litz, and Joseph Devietti. Rpg2: Robust profile-guided runtime prefetch generation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 999–1013, 2024.

[46] Kai Zhang and Chao Tian. Fundamental limits of coded caching: From uncoded prefetching to coded prefetching, 2017.

[47] Agustín Navarro-Torres, Biswabandan Panda, Jesús Alastruey-Benedé, Pablo Ibáñez, Víctor Viñals-Yúfera, and Alberto Ros. Berti: an accurate local-delta data prefetcher. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 975–991. IEEE, 2022.

[48] Qian Yu, Mohammad Ali Maddah-Ali, and A. Salman Avestimehr. Characterizing the rate-memory tradeoff in cache networks within a factor of 2, 2018.

[49] David Monniaux and Valentin Touzeau. On the complexity of cache analysis for different replacement policies, 2019.

[50] Shaahin Angizi, Navid Khoshavi, Andrew Marshall, Peter Dowben, and Deliang Fan. Meram: Non-volatile cache memory based on magneto-electric fets, 2020.

[51] N. Ramasubramanian, Srinivas V. V., and N. Ammasai Gounden. Performance of cache memory subsystems for multicore architectures, 2011.

[52] Irina Calciu, M Talha Imran, Ivan Puddu, Sanidhya Kashyap, Hasan Al Maruf, Onur Mutlu, and Aasheesh Kolli. Rethinking software runtimes for disaggregated memory. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 79–92, 2021.

[53] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention, 2023.

[54] Atanu Barai, Gopinath Chennupati, Nandakishore Santhi, Abdel-Hameed A. Badawy, and Stephan Eidenbenz. Modeling shared cache performance of openmp programs using reuse distance, 2019.

[55] Yuhang Liu, Luming Wang, Xiang Li, Yang Wang, Mingyu Chen, and Yungang Bao. Genepatterns: Should architecture be customized for each application?, 2019.

[56] Satya N. Majumdar and Jaikumar Radhakrishnan. Analytical studies of strategies for utilization of cache memory in computers, 2000.

[57] Daniel S Berger, Ramesh K Sitaraman, and Mor Harchol-Balter. {AdaptSize}: Orchestrating the hot object memory cache in a content delivery network. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 483–498, 2017.

[58] Hariharan Devarajan, Anthony Kougkas, and Xian-He Sun. Hfetch: Hierarchical data prefetching for scientific workflows in multi-tiered storage environments. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 62–72. IEEE, 2020.

[59] Julian Hammer, Georg Hager, Jan Eitzinger, and Gerhard Wellein. Automatic loop kernel analysis and performance modeling with kerncraft, 2015.

[60] P. Jayarekha and T. R. Gopalakrishnan Nair. An adaptive dynamic replacement approach for a multicast based popularity aware prefix cache memory system, 2010.

[61] Vajira Thambawita, Roshan G. Ragel, and Dhammike Elkaduwe. To use or not to use: Cpus' cache optimization techniques on gpgpus, 2018.

[62] Navid Khoshavi, Xunchao Chen, Jun Wang, and Ronald F. DeMara. Read-tuned stt-ram and edram cache hierarchies for throughput and energy enhancement, 2016.

[63] Exploiting vector code semantics.

[64] 2019 ieee international symposiu.

[65] Oskar Schirmer. Parallel architecture hardware and general purpose operating system co-design, 2018.

[66] Jérémie Lagravière, Johannes Langguth, Martina Prugger, Lukas Einkemmer, Phuong H. Ha, and Xing Cai. Performance optimization and modeling of fine-grained irregular communication in upc, 2019.

[67] Crysttian Arantes Paixão and Flávio Codeço Coelho. Computable compressed matrices, 2013.

[68] Alireza Farshin, Amir Roozbeh, Gerald Q Maguire Jr, and Dejan Kostić. Reexamining direct cache access to optimize {I/O} intensive applications for multi-hundred-gigabit networks. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 673–689, 2020.

[69] Xiang Zhang, Kai Wan, Hua Sun, Mingyue Ji, and Giuseppe Caire. A new design of cache-aided multiuser private information retrieval with uncoded prefetching, 2021.

[70] Josefa Díaz Álvarez, José L. Risco-Martín, and J. Manuel Colmenar. Evolutionary design of the memory subsystem, 2023.

[71] Julian Hammer, Jan Eitzinger, Georg Hager, and Gerhard Wellein. Kerncraft: A tool for analytic performance modeling of loop kernels, 2017.

[72] Nitish Mital, Deniz Gunduz, and Cong Ling. Coded caching in multi-server system with random topology, 2020.

[73] Lukas Trümper, Tal Ben-Nun, Philipp Schaad, Alexandru Calotoiu, and Torsten Hoefler. Performance embeddings: A similarity-based approach to automatic performance optimization, 2023.

[74] Neelu Shivprakash Kalani and Biswabandan Panda. Instruction criticality based energy-efficient hardware data prefetching. *IEEE Computer Architecture Letters*, 20(2):146–149, 2021.

[75] Ravi Khatwal and Manoj Kumar Jain. Application specific cache simulation analysis for application specific instruction set processor, 2014.

[76] Hao Wu, Fangfei Liu, and Ruby B. Lee. Cloud server benchmarks for performance evaluation of new hardware architecture, 2016.

[77] Vinson Young, Aamer Jaleel, Evgeny Bolotin, Eiman Ebrahimi, David Nellans, and Oreste Villa. Combining hw/sw mechanisms to improve numa performance of multi-gpu systems. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 339–351. IEEE, 2018.

[78] Keren Zhou, Xiaozhu Meng, Ryuichi Sai, and John Mellor-Crummey. Gpa: A gpu performance advisor based on instruction sampling, 2020.

[79] Majid Jalili and Mattan Erez. Reducing load latency with cache level prediction, 2021.

[80] Johannes Hofmann, Georg Hager, and Dietmar Fey. On the accuracy and usefulness of analytic energy models for contemporary multicore processors, 2018.

[81] Christie Alappat, Nils Meyer, Jan Laukemann, Thomas Gruber, Georg Hager, Gerhard Wellein, and Tilo Wettig. Ecm modeling and performance tuning of spmv and lattice qcd on a64fx, 2021.

AI-generated, for reference only.