Advanced Computing Concepts in Asynchronous Circuits and Processor Design: A Survey

www.surveyx.cn

Abstract

This survey paper presents a comprehensive exploration of advanced computing concepts and their transformative impact on processor design. Key areas of focus include asynchronous circuits, which offer significant power efficiency and performance benefits over traditional synchronous designs, and superscalar processors, which enhance computational throughput through instruction-level parallelism. The paper also examines parallel computing frameworks that optimize large-scale data processing and the open-source RISC-V architecture, which enables customization and innovation in processor design. Event-driven processing is highlighted for its role in improving energy efficiency and responsiveness, particularly in high-energy physics and spiking neural networks. The survey underscores the necessity of integrating advanced computing concepts to address modern computational challenges and drive future innovations in processor architecture. Future research directions emphasize refining models for asynchronous circuits, exploring sustainable design approaches, and enhancing parallel and distributed computing education. The findings highlight the critical role of these concepts in advancing computational efficiency and performance, paving the way for more resilient and scalable computing systems.

1 Introduction

1.1 Importance of Advanced Computing Concepts

Advanced computing concepts are crucial in modern technology, significantly shaping processor design and functionality to meet the demands of an increasingly computationally intensive landscape [1]. Asynchronous circuits, for example, present a viable alternative to traditional synchronous designs, enhancing power efficiency and performance, which are essential for size-weight-and-power (SWaP)-constrained applications [2]. This transition is particularly pertinent in high-performance computing (HPC), where optimizing power systems is vital for achieving net-zero emissions [3].

The integration of instruction-level parallelism (ILP) and thread-level parallelism (TLP) in processor designs has been pivotal in addressing the rising demand for high-performance embedded processors in consumer electronics. However, these strategies encounter limitations, prompting the need for innovative methodologies to enhance processor capabilities [4]. The adoption of asynchronous circuits is further complicated by insufficient support from existing hardware generation languages (HGLs), indicating a necessity for improved language features to facilitate their broader implementation [5].

Moreover, the rise of parallel computing has transformed large-scale data processing, with asynchronous parallel computing methods proving essential for solving complex problems characterized by nonseparable linear constraints [6]. This development is particularly relevant given digital computing's challenges in efficiently processing continuous analogue information [7]. The investigation of advanced computing concepts, such as the Global Cellular Automata (GCA) model, highlights the adaptability of contemporary computational frameworks to evolving technological demands [8].

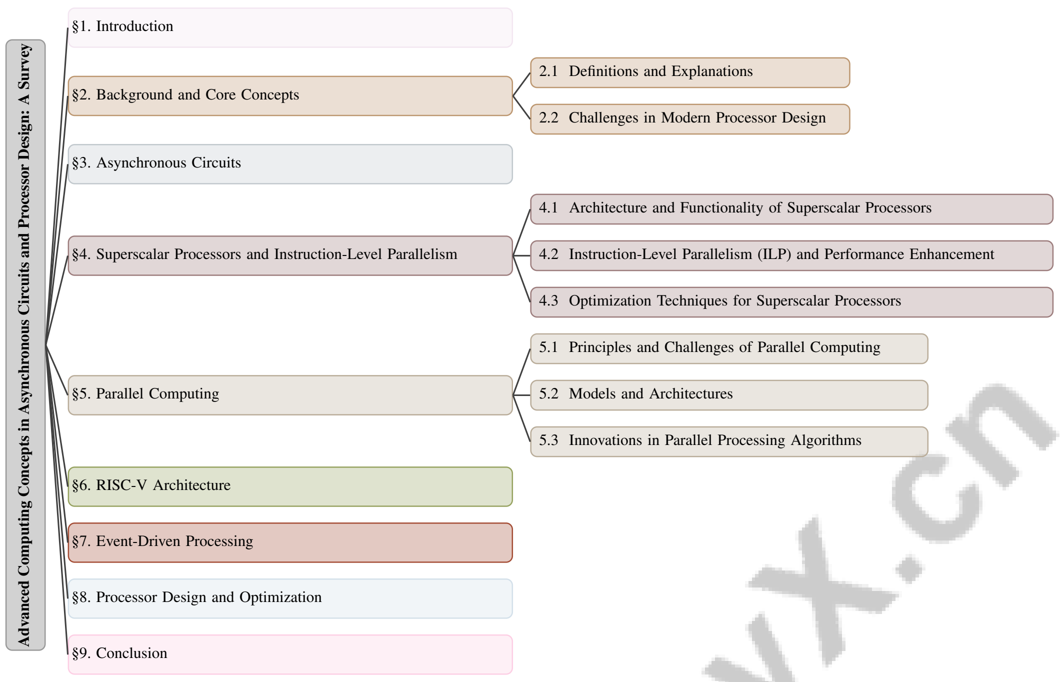

Advanced Computing Concepts in Asynchronous Circuits and Processor Design: A Survey
§1. Introduction
§2. Background and Core Concepts
2.1 Definitions and Explanations
2.2 Challenges in Modern Processor Design
§3. Asynchronous Circuits
§4. Superscalar Processors and Instruction-Level Parallelism
4.1 Architecture and Functionality of Superscalar Processors
4.2 Instruction-Level Parallelism (ILP) and Performance Enhancement
4.3 Optimization Techniques for Superscalar Processors
§5. Parallel Computing
5.1 Principles and Challenges of Parallel Computing
5.2 Models and Architectures
5.3 Innovations in Parallel Processing Algorithms
§6. RISC-V Architecture
§7. Event-Driven Processing
§8. Processor Design and Optimization
§9. Conclusion

Figure 1: chapter structure

1.2 Structure of the Survey

This survey provides an in-depth exploration of advanced computing concepts and their implications for processor design. It begins with an introduction that emphasizes the importance of these concepts in contemporary technology, followed by a comprehensive background section defining key terms such as asynchronous circuits, superscalar processors, parallel computing, instruction-level parallelism, RISC-V architecture, event-driven processing, and processor design. The survey then transitions into specific topics, starting with asynchronous circuits and their advantages and applications in processor design.

Subsequent sections analyze the architecture and functionality of superscalar processors, focusing on the role of instruction-level parallelism in performance enhancement. The survey further investigates principles of parallel computing, various models, architectures, and recent advancements in parallel processing algorithms. It introduces the RISC-V architecture, underscoring its influence on processor design and customization. Additionally, event-driven processing is examined, particularly in the context of high-energy physics and spiking neural networks.

The survey concludes by addressing processor design and optimization techniques, illustrating how advanced computing concepts are integrated to enhance computational efficiency and performance. The final section synthesizes key findings and proposes future directions for advancements in processor design.The following sections are organized as shown in Figure 1.

2 Background and Core Concepts

2.1 Definitions and Explanations

Asynchronous circuits operate without a global clock, relying on local handshaking protocols for data transfers and state transitions, enhancing timing flexibility and offering potential power savings crucial for energy-efficient hardware design [9, 10]. These circuits are modeled through multi-valued functions that map multi-dimensional inputs to outputs, necessitating refined models for circuits with multiple clock domains [11].

Superscalar processors enhance computational throughput by executing multiple instructions per cycle, leveraging instruction-level parallelism (ILP) to optimize execution unit utilization and overall performance [12]. Parallel computing, through various models and architectures, efficiently manages

AI-generated, for reference only.

large-scale computations. The Global Cellular Automata (GCA) model, for instance, provides a flexible framework enabling dynamic cell communication, addressing classical cellular automata limitations [8].

The RISC-V architecture, an open-source instruction set, simplifies processor design and customization, significantly influencing modern processor architectures [12]. Event-driven processing, which responds to state changes rather than following a set sequence, enhances energy efficiency, proving beneficial in high-energy physics and spiking neural networks (SNNs) where minimizing physical time impact is crucial [13].

Processor design focuses on optimizing and integrating components to improve efficiency and performance, addressing challenges such as integrated circuits' vulnerability to physical attacks [14]. Concepts like computational memory (CM), utilizing phase-change memory devices for computation at the data storage location, are pivotal in optimizing processor performance by reducing data transfer delays.

2.2 Challenges in Modern Processor Design

The transition from clock frequency scaling to parallel processing paradigms introduces challenges requiring innovative strategies to enhance computational efficiency and performance. A key challenge is the non-deterministic behavior of asynchronous systems, complicating system behavior prediction and verification over time [10]. Ensuring deadlock-free operation in asynchronous circuits is difficult due to the lack of a comprehensive mathematical framework for asynchronous automata [15]. Inadequate classification of sequential circuits, especially with multiple clock domains, leads to inefficiencies in circuit design [11].

In multi-core processor design, there is a significant misalignment between current programming paradigms and modern architectures, resulting in suboptimal parallel execution unit utilization [16]. This is compounded by the challenges of designing efficient out-of-order execution systems that maintain high clock speeds while managing microarchitectural complexity [17]. Current multiprocessor scheduling methods for directed acyclic graph (DAG) tasks are inefficient, as cache memory adversely affects task schedulability by increasing worst-case execution times (WCET) [18].

Mapping data parallel computations onto the memory hierarchy of modern multi-core systems remains challenging, leading to poor cache utilization and performance losses [19]. This inefficiency is exacerbated by resource competition in parallel computing systems, causing congestion and reducing performance [20]. Existing ranking and selection (RS) methods struggle with limited systems, particularly when alternatives exceed a few hundred, due to the need for statistical validity [21].

The heat wall limits traditional digital computers' clock speed, rendering them inadequate for real-time analogue data processing [7]. This necessitates developing scalable and efficient parallel computing frameworks capable of managing burst-parallel jobs on serverless platforms while minimizing data movement and maximizing resource utilization [22]. However, existing methods face excessive scheduling overhead and data movement challenges, leading to performance bottlenecks and increased costs [22].

Reliance on synchronous operations introduces security vulnerabilities, inadequately detecting or mitigating physical attacks on integrated circuits [14]. Addressing these challenges requires developing innovative methodologies and tools prioritizing adaptability and resource optimization, facilitating the transition towards more efficient and scalable parallel processing paradigms in modern processor design. Such advancements are crucial for enhancing computational systems' performance and reliability amid escalating computational demands.

In recent years, the exploration of asynchronous circuits has gained significant momentum within the field of digital design. These circuits offer several advantages over their synchronous counterparts, particularly in terms of performance and power efficiency. The hierarchical structure of asynchronous circuits, as depicted in Figure 2, illustrates these benefits clearly. This figure not only highlights the inherent advantages of asynchronous designs, such as enhanced performance and flexibility, but also emphasizes their applications in processor design, including integration within micropipeline architectures and the optimization of computational processes. By examining this structure, we can better understand the potential of asynchronous circuits to revolutionize current design methodologies and improve overall system efficiency.

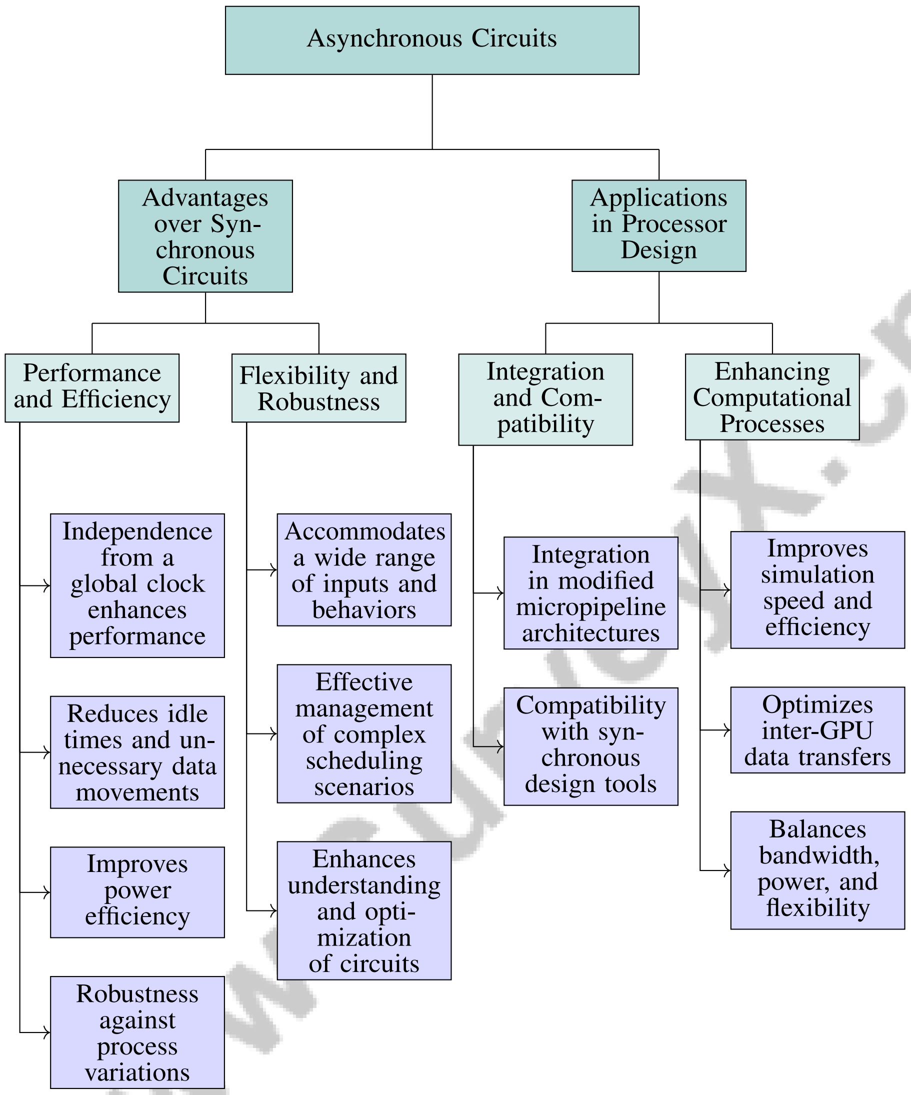

Asynchronous Circuits
Advantages over Synchronous Circuits
Applications in Processor Design
Performance and Efficiency
Flexibility and Robustness
Integration and Compatibility
Enhancing Computational Processes
Independence from a global clock enhances performance
Reduces idle times and unnecessary data movements
Improves power efficiency
Robustness against process variations
Accommodates a wide range of inputs and behaviors
Effective management of complex scheduling scenarios
Enhances understanding and optimization of circuits
Integration in modified micropipeline architectures
Compatibility with synchronous design tools
Improves simulation speed and efficiency
Optimizes inter-GPU data transfers
Balances bandwidth, power, and flexibility

Figure 2: This figure illustrates the hierarchical structure of asynchronous circuits, highlighting their advantages over synchronous circuits and applications in processor design. Key benefits include enhanced performance, power efficiency, and flexibility, while applications emphasize integration in micropipeline architectures and optimization of computational processes.

3 Asynchronous Circuits

3.1 Advantages over Synchronous Circuits

Asynchronous circuits present several advantages over their synchronous counterparts, primarily due to their independence from a global clock, which allows autonomous operation of processing elements. This autonomy optimizes local latencies and enhances system performance by eschewing global worst-case limitations, thereby reducing idle times and unnecessary data movements to improve power efficiency, especially in energy-sensitive applications [14]. The application of dynamical systems concepts in modeling asynchronous circuits enhances analytical capabilities, providing a robust theoretical foundation for design and performance evaluation [23]. This approach facilitates a comprehensive understanding of stability in asynchronous systems, crucial for managing complex

AI-generated, for reference only.

circuit behaviors [10]. Moreover, formalization through generator functions and analytical tools contributes to more efficient and reliable asynchronous system designs.

Asynchronous circuits demonstrate robustness against process variations, ensuring performance stability across diverse conditions [10]. Their flexibility accommodates a wide range of inputs and behaviors, creating a versatile framework for various computational tasks [24]. Simulation capabilities, such as those provided by PyHGL, enhance power efficiency and performance through high code density and efficient processes [5]. These circuits manage complex scheduling scenarios effectively, as shown by frameworks like EBSP, which yield more efficient outcomes than simpler heuristics [25]. Additionally, methods like Causal Function Classification (CFC) improve the understanding and optimization of synchronous and multiple clock domain circuits [11].

The inherent advantages of asynchronous circuits, including power efficiency, performance, and flexibility, make them a superior choice for modern processor design, addressing contemporary computing challenges and driving advancements in computational efficiency [7].

3.2 Applications in Processor Design

Asynchronous circuits are crucial in modern processor design, enhancing computational efficiency and flexibility. A key application is the integration of asynchronous control logic within modified micropipeline architectures, maintaining compatibility with synchronous design tools and facilitating broader adoption [26]. This integration allows seamless incorporation of asynchronous principles into existing frameworks, improving versatility and performance.

In VLSI circuit simulation, the One-Pass Waveform-Based GPU Parallelism (OPW-GPU) method exemplifies asynchronous processing application, significantly enhancing simulation speed and efficiency [27]. This underscores the capacity of asynchronous circuits to enhance computational processes, especially in complex simulation environments. Asynchronous circuits also optimize inter-GPU data transfers, as demonstrated by frameworks like Ripple, which provide a unified view of computational space across multiple dimensions and GPUs, facilitating optimal data transfer scheduling and enhancing computational efficiency in large-scale systems [28].

Furthermore, asynchronous circuits play a critical role in balancing bandwidth, power, and flexibility in routing methodologies, vital for scalable and efficient processor design [29]. This balance is essential in resource-optimized systems, ensuring processor designs meet contemporary computing demands. The exploration of asynchronous concepts such as controllability and accessibility provides a framework for enhancing robustness and adaptability in processor designs [10]. This supports the development of reliable and efficient asynchronous systems adaptable to diverse computational tasks and environments.

Additionally, asynchronous circuits optimize serverless parallel computing frameworks like Wukong, using decentralized scheduling to enhance performance and resource efficiency [22]. By enabling parallel scheduling and execution of tasks while maintaining data locality, Wukong exemplifies the potential of asynchronous circuits to drive innovations in parallel computing methodologies.

The integration of asynchronous circuits in processor design marks a significant advancement in modern computing, enhancing flexibility and efficiency through event-driven operation, minimizing power consumption during idle periods, and exhibiting robustness against variations in process, voltage, and temperature (PVT). This adaptability facilitates the development of low-power digital circuits and supports energy-efficient systems tailored for emerging applications, such as wireless sensor networks and portable medical devices, broadening the scope and performance of contemporary computing architectures [26, 30, 1, 12]. These contributions underscore the transformative potential of asynchronous circuits in addressing contemporary computational challenges and driving future innovations in processor architecture.

As illustrated in Figure 3, which categorizes key applications of asynchronous circuits in processor design, these circuits play a vital role in enhancing control logic, simulation efficiency, and routing flexibility. The figure highlights three main areas: the integration of asynchronous control logic, improvements in VLSI circuit simulation and inter-GPU data transfer, and the balance of bandwidth and flexibility in routing methodologies. The visual examples within the figure further emphasize the transformative role of asynchronous circuits in advancing processor design by streamlining operations and enhancing computational efficiency [31, 32].

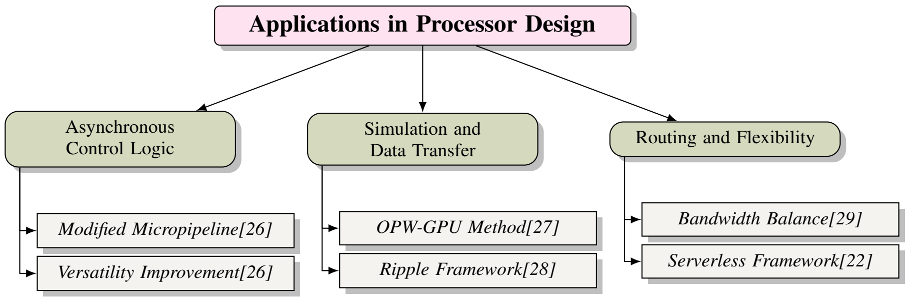

Applications in Processor Design
Asynchronous Control Logic
Simulation and Data Transfer
Routing and Flexibility
Modified Micropipeline[26]
Versatility Improvement[26]
OPW-GPU Method[27]
Ripple Framework[28]
Bandwidth Balance[29]
Serverless Framework[22]

Figure 3: This figure illustrates key applications of asynchronous circuits in processor design, highlighting their roles in enhancing control logic, simulation efficiency, and routing flexibility. The figure categorizes these applications into three main areas: integration of asynchronous control logic, improvements in VLSI circuit simulation and inter-GPU data transfer, and the balance of bandwidth and flexibility in routing methodologies.

4 Superscalar Processors and Instruction-Level Parallelism

4.1 Architecture and Functionality of Superscalar Processors

Superscalar processors represent a pivotal advancement in microarchitecture, enabling the execution of multiple instructions per cycle through the exploitation of instruction-level parallelism (ILP). This is achieved via sophisticated pipeline architectures that facilitate parallel dispatch and execution, thereby optimizing throughput and resource utilization [33]. Central to their design are multiple execution units that support concurrent processing, enhancing computational performance [17]. A hallmark of these processors is out-of-order execution, which dynamically schedules instructions based on data dependencies and resource availability, thus reducing latency and maximizing throughput [17]. This transition introduces complexity in maintaining clock frequency and managing intricate logic [17].

The verification of RISC-V superscalar processors is crucial for ensuring correctness in complex designs [34]. Integration of advanced scheduling algorithms, such as initialization heuristics and Integer Linear Programming (ILP) methods, further optimizes performance [25]. Techniques like branch prediction and memory-level parallelism (MLP) profiling significantly enhance processing efficiency. The analytical MLP model, which profiles workload memory behavior to estimate MLP across architectures, contributes to performance improvements [35]. Additionally, frameworks like PyHGL enable the integration of asynchronous circuits within superscalar architectures, enhancing design versatility [5].

The pursuit of energy efficiency in superscalar microarchitectures is aligned with leveraging ILP while minimizing energy consumption, addressing the demand for sustainable computing solutions [33]. This balance is essential in modern environments requiring high-performance processors that are both powerful and energy-efficient. Superscalar processors, characterized by their ability to execute multiple instructions per cycle through advanced scheduling and architectural innovations, optimize computational tasks across diverse domains, including high-performance computing, machine learning, and AI, solidifying their foundational role in modern systems [36, 12, 37, 38].

As illustrated in Figure 4, which depicts the key components of superscalar processor architecture, instruction-level parallelism is crucial for enabling simultaneous execution of multiple instructions within a single clock cycle. The figure highlights various execution techniques, including out-of-order execution, which are essential for maximizing throughput. Performance optimization strategies, such as advanced scheduling and energy efficiency, are also emphasized, alongside verification and design considerations pertinent to RISC-V verification and the application of queuing models. The "Instruction Execution in a Single Clock Cycle" example demonstrates concurrent processing of instructions, optimizing the cycle time for enhanced speed. The "Sequential Processing of Data Streams" example further elucidates operational dynamics, showcasing how data streams can be efficiently processed through sequential operations. Together, these examples highlight the advanced capabilities of superscalar processors in leveraging ILP to optimize computing performance [12, 32].

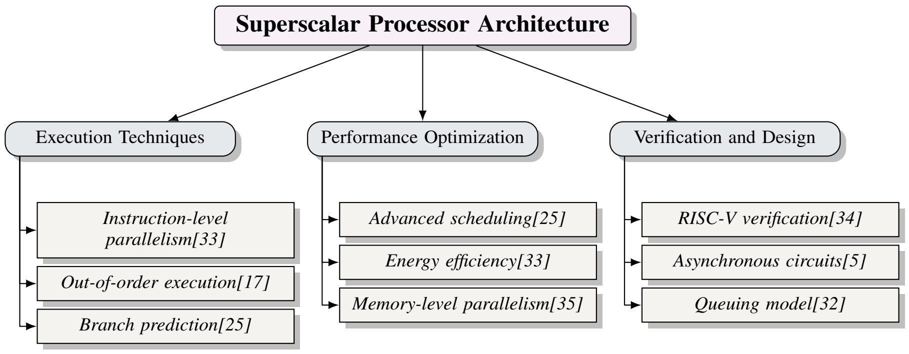


Figure 4: This figure illustrates the key components of superscalar processor architecture, highlighting execution techniques, performance optimization strategies, and verification and design considerations. Execution techniques such as instruction-level parallelism and out-of-order execution are crucial for maximizing throughput. Performance optimization focuses on advanced scheduling and energy efficiency, while verification and design include RISC-V verification and the use of queuing models.

4.2 Instruction-Level Parallelism (ILP) and Performance Enhancement

Instruction-Level Parallelism (ILP) is integral to enhancing processor performance by allowing the concurrent execution of non-dependent instructions, optimizing throughput and resource utilization. This is achieved through advanced pipeline architectures that enable parallel dispatch and execution [17]. ILP in superscalar processors is further augmented by sophisticated instruction scheduling architectures, such as fused-logic matrix schedulers, which improve performance in soft processors [39].

Speculation techniques, including value prediction and load-address prediction, enhance ILP by anticipating operand values and load addresses, maintaining high levels of parallel execution despite data dependencies [32]. In heterogeneous environments, optimization techniques like Layer-Based Partitioning (LBP) minimize communication overhead, ensuring efficient data transfer and maintaining high ILP levels [40]. The development of a warp scheduler that circumvents traditional superscalar issues using a round-robin method enhances instruction distribution and ILP efficiency [41].

Advanced methodologies, such as SupeRFIVe, use instruction set simulators (ISS) to validate superscalar processors, synchronizing them with hardware verification language (HVL) testbenches for robust performance [34]. Additionally, the C-slow technique allows a single-core processor to execute multiple threads in parallel, improving throughput without the complexity of traditional multithreading architectures [4].

The analytical MLP model significantly reduces prediction error for total time waiting for DRAM, achieving an average absolute error of 3.6

Furthermore, the FMC method supports massively parallel computing by processing continuous-time signals represented by various sinusoidal signals [7]. This capability underscores ILP's transformative impact on computational performance, enabling detailed analyses that were previously infeasible.

The ongoing advancement of ILP techniques, coupled with their strategic integration into processor architectures, is essential for meeting the escalating demands of high-performance computing (HPC) environments. This evolution enhances the efficiency of parallel computing methods—such as multithreading on CPUs and the use of specialized accelerators like GPUs and FPGAs—and supports innovative architectures like invasive tightly coupled processor arrays (TCPAs) that prioritize resource awareness and adaptability. As the complexity of parallel algorithms and hardware architectures increases, optimizing these systems becomes vital for overcoming performance bottlenecks and achieving scalability, ensuring HPC infrastructures can meet diverse scientific and industrial application requirements [38, 42, 37, 43, 36]. By enabling concurrent instruction execution, ILP maximizes resource utilization and enhances computational throughput, driving advancements in modern processor architectures.

4.3 Optimization Techniques for Superscalar Processors

Superscalar processors have achieved remarkable performance enhancements through various optimization techniques addressing architectural and operational challenges. A significant improvement is the partitioning of issue and bypass logic into multiple clusters, enabling faster clock speeds compared to traditional single-window architectures [17]. This clustering reduces issue logic complexity, allowing more efficient instruction dispatch and execution.

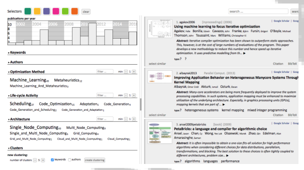
Advanced prediction mechanisms, such as the VSEP predictor and LAPELE mechanism, optimize instruction dependencies and memory access patterns, reducing execution latency by capturing previously unexploited value patterns and enabling early load execution [12]. Domain-specific optimizations, like those in the COMET programming language for tensor contractions, facilitate efficient execution of high-level expressions on architecture-specific hardware, further optimizing computational tasks [43].

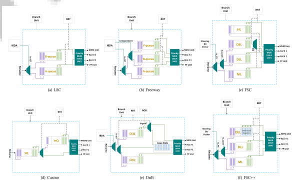
The queuing model for CPU functional units, which models the issue queue and functional units as a queuing network, provides insights into expected queue lengths for various instruction types and configurations, aiding efficient instruction scheduling and resource allocation [32]. The use of CUDA in GPGPU processing demonstrates significant performance gains in applications requiring parallel processing capabilities, such as video processing and scientific computations [44].

Dynamic scheduling and speculative execution, as illustrated by the SPSim framework, allow concurrent instruction execution beyond traditional pipelining constraints, effectively addressing data hazards and enhancing processor performance [45]. The GADGET algorithm optimizes resource allocation for RAR-based DDL jobs through temporal submodularity, further improving superscalar efficiency [46].

The fused-logic matrix scheduler integrates wakeup and selection logic into a single circuit, streamlining instruction scheduling and enhancing speed and efficiency [39]. Implementing wide-issue clusters with a simple Mod-N steering policy optimizes load balancing and minimizes inter-cluster communication, further enhancing superscalar performance [33].

Finally, the C-slow technique introduces a novel approach by replacing each register with multiple registers, enabling simultaneous execution of computations and increasing throughput without the complexity of traditional multithreading architectures [4]. Collectively, these optimization techniques significantly advance the capabilities of superscalar processors, ensuring their continued effectiveness in high-performance computing environments.


(a) Research in Machine Learning and Optimization Methods[47]


(b) Processor Architecture Diagrams[48]

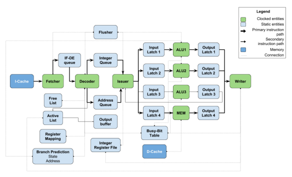

(c) Instruction Fetch and Decode Pipeline[45]

Figure 5: Examples of Optimization Techniques for Superscalar Processors

As shown in Figure 5, superscalar processors are designed to execute multiple instructions during a single clock cycle by leveraging instruction-level parallelism. The optimization techniques employed are crucial for enhancing efficiency and performance. The first figure, "Research in Machine Learning and Optimization Methods," highlights the intersection of these fields in advancing processor capabilities. The second figure, "Processor Architecture Diagrams," illustrates how data and instructions flow through systems like LSC, Freeway, and FSC++, utilizing priority queues and issue units to optimize processing. Finally, the "Instruction Fetch and Decode Pipeline" image details the stages from I-Cache to decoding, essential for maintaining high throughput in superscalar processors. Together, these examples underscore the intricate optimization strategies driving modern superscalar processor performance [47, 48, 45].

5 Parallel Computing

5.1 Principles and Challenges of Parallel Computing

Parallel computing serves as a cornerstone of modern computational science, facilitating the concurrent execution of multiple tasks to improve processing efficiency and manage extensive data operations. The transition from single-core to multi-core and manycore processors has driven the evolution of parallel computing, necessitating sophisticated models and frameworks [16]. This evolution underscores parallel computing's integration into commodity hardware, marking its vital role in advancing computer architecture [49].

Central to parallel computing are models like the Parallel Random Access Machine (PRAM), Bulk Synchronous Parallel (BSP), and DataFlow models, which provide frameworks for parallel execution [50]. These models are crucial for task parallelization, accommodating varied communication intensities and computational demands, especially in heterogeneous multicore platforms [49]. Figure 6 illustrates the key principles, challenges, and innovative approaches in parallel computing. It highlights foundational models such as PRAM, BSP, and DataFlow, addresses challenges like multithreading inefficiencies and resource management, and showcases innovative solutions including MapReduce algorithms and OpenMP translation.

Despite its benefits, parallel computing faces challenges, notably the complexity and inefficiency of multithreading models, which often lack a robust theoretical foundation [50]. The effective utilization of inter-thread cache benefits to reduce worst-case execution times (WCET) and improve high-utilization task schedulability presents additional hurdles [18]. Viewing parallel computing processes as players in a congestion game further highlights the complexities of resource allocation and its impact on execution time [20]. This necessitates efficient resource management strategies to mitigate performance bottlenecks. Furthermore, simplifying parallel programming in low-energy platforms like GAP8 remains challenging [51].

Innovative approaches, such as MapReduce algorithms, offer solutions to parallel computation complexities, drawing comparisons to combinational Boolean circuits [52]. These methodologies enhance computational power and efficiency, addressing limitations of traditional parallel computing.

Parallel computing holds substantial potential for enhancing computational performance by enabling multiple processors to collaborate on complex tasks. However, it introduces challenges such as optimizing parallel algorithms and managing data transfer in heterogeneous systems, requiring continuous research to fully leverage its capabilities in high-performance computing environments [36, 53, 54, 38]. Developing new trends in parallel and distributed simulation techniques, as well as exploring cache-aware parallel programming strategies, is essential for advancing the field.

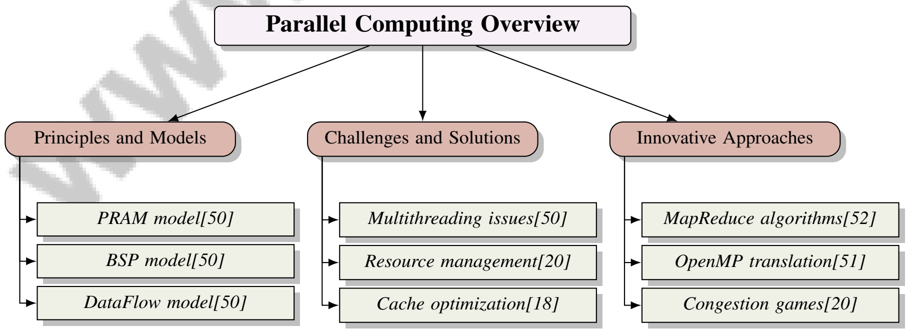


Figure 6: This figure illustrates the key principles, challenges, and innovative approaches in parallel computing. It highlights foundational models such as PRAM, BSP, and DataFlow, addresses challenges like multithreading inefficiencies and resource management, and showcases innovative solutions including MapReduce algorithms and OpenMP translation.

5.2 Models and Architectures

Parallel computing encompasses diverse models and architectures designed to enhance computational tasks by distributing workloads across multiple processing units, including multithreaded CPUs,

AI-generated, for reference only.

GPUs, and specialized architectures like SIMD. This approach is crucial for high-performance computing (HPC), enabling effective partitioning of large problems into manageable tasks. However, optimizing parallel program execution is complex due to challenges like load balancing, data transfer bottlenecks in heterogeneous systems, and the need for sophisticated software optimization techniques, potentially involving machine learning and heuristic methods. Additionally, the increasing core count in modern computing necessitates understanding performance metrics, scalability, and the relationship between computational and communication complexities in large-scale systems [55, 38, 47, 56, 36].

The Parallel Random Access Machine (PRAM) model provides a theoretical framework for parallel algorithm design, assuming shared memory architecture. It aids in conceptualizing parallel execution and exploring memory architectures' impact on efficiency [50]. Architectures like multi-core CPUs, GPUs, FPGAs, and distributed systems each offer unique advantages and challenges. Multi-core CPUs and GPUs efficiently handle concurrent tasks, FPGAs provide customizable hardware solutions, and distributed systems enable large-scale computations through interconnected nodes.

The T10 method exemplifies optimizing execution by partitioning Deep Neural Network (DNN) computation across cores, utilizing high-bandwidth inter-core communication. This highlights the importance of optimizing data movement and communication in parallel architectures, especially where interprocessor communication poses bottlenecks. Efficient management of these pathways is essential for maximizing benefits in multi-threaded and distributed environments [57, 55, 38, 56, 36]. Hybrid computing models combining CPU and GPU computation enhance performance and flexibility.

Process-oriented programming treats classes as processes, enabling communication and method execution across nodes with minimal syntax changes, facilitating scalable applications. Asynchronous parallel computing algorithms reduce execution time compared to synchronous methods, achieving speedups while maintaining stability, highlighting their potential in enhancing parallel performance [58].

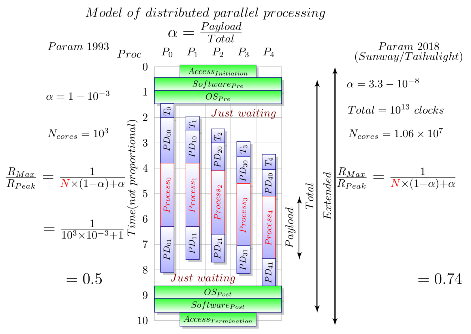
Hybrid control system prototypes linking real-time control systems with parallel computing farms integrate parallel computing with real-time applications, crucial for high computational power and real-time responsiveness. Spiking Neural Networks (SNNs) face limitations due to sequential computational dependency, hampering efficiency in parallel environments, necessitating solutions for advancing SNN capabilities [58].

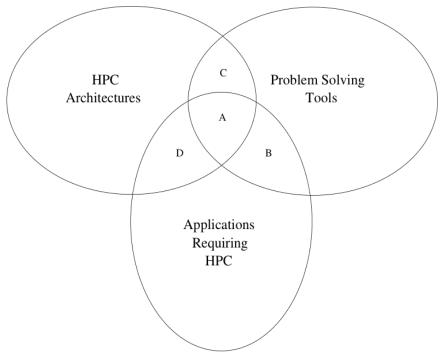
Exploring parallel computing models and architectures reveals strategies to improve computational efficiency, including multithreading on CPUs, integrating GPUs and accelerators, and using specialized architectures like SIMD. This exploration underscores challenges in optimizing parallel programs and addressing bottlenecks, particularly in heterogeneous systems, while emphasizing understanding speedup and efficiency metrics, as outlined by laws such as Amdahl's and Gustafson's. By synthesizing these insights, researchers can navigate parallel computing complexities, advancing towards effective high-performance computing solutions [36, 38]. Understanding benefits and challenges of each model aids in developing solutions for modern computing demands.

As depicted in Figure 7, understanding parallel computing's models and architectures is crucial. The first image, a model of distributed parallel processing, illustrates task distribution across processes, emphasizing time and payload distribution for efficient computation. This model highlights task division and synchronization to minimize processing time, fundamental for optimal parallel computing performance. The second image, a Venn diagram, delineates relationships between HPC Architectures, Problem Solving Tools, and Applications requiring HPC, serving as a conceptual map illustrating intersections addressing complex computational problems. These visual representations provide foundational understanding of organizational and operational principles driving parallel computing, paving the way for further exploration into its applications and advancements [59, 56].

5.3 Innovations in Parallel Processing Algorithms

Recent innovations in parallel processing algorithms have significantly advanced computational efficiency and scalability, particularly in high-performance computing (HPC) environments. Modular approaches simplify complex programming tasks by explicitly managing parallelism and avoiding resource contention, enhancing robustness and scalability of parallel processing frameworks [50].


(a) Model of distributed parallel processing[59]



(b) Venn Diagram of HPC Architectures, Problem Solving Tools, and Applications Requiring HPC[56]

Figure 7: Examples of Models and Architectures

Frameworks analyzing resource usage in parallel logic programs without execution represent significant advancements, providing resource usage bounds for efficient program design and optimization [49]. Game-theoretical approaches for analyzing parallel algorithms' efficiency suggest further study on optimality and equilibrium principles to enhance performance [20].

In multi-core architectures, future research should develop better abstractions and new programming paradigms to leverage system capabilities [16]. Simplified OpenMP for GAP8 shows significant code size reduction and decreased programming workload, facilitating parallel computing in low-energy IoT devices [51]. This highlights parallel computing's potential to enhance efficiency and reduce energy consumption in emerging technologies.

Strengthening the relationship between NC and deterministic MapReduce hierarchies, proving any NC problem solvable by MapReduce in O(T(n)/ log n) rounds, underscores exploring new parallelization strategies and comprehensive studies integrating optimization techniques to enhance computational power and efficiency [52].

Continuous development of parallel processing techniques is essential for addressing modern computing demands. Innovations in parallel processing algorithms improve efficiency and scalability, enabling effective solutions for complex tasks. Progress emphasizes performance analysis and speedup laws like Amdahl's and Gustafson's to understand and optimize parallel algorithms' behavior. Advanced programming techniques, including multithreading, GPU acceleration, and meta-heuristic approaches, enhance performance across heterogeneous systems, addressing data transfer and algorithm complexity bottlenecks, paving the way for future high-performance computing advancements [36, 38, 47, 21]. Future research should optimize algorithms for specific hardware, explore new parallel computing applications, and address dynamic neighborhood management challenges.

6 RISC-V Architecture

6.1 Impact on Processor Design

The RISC-V architecture marks a significant transformation in processor design, characterized by its open-source nature and extensive customization capabilities. As an open-source instruction set architecture (ISA), RISC-V allows for tailored processor designs, fostering a collaborative ecosystem that enhances computational efficiency across various fields, including high-performance computing and machine learning. Recent advancements, like dual-issue superscalar processors with dynamic execution and custom vector instruction sets, underscore RISC-V's potential to improve instruction throughput and energy efficiency [4, 37, 60, 34, 61].

Enhancements in RISC-V processor design, such as the integration of Single Instruction, Multiple Data (SIMD) and dynamic execution capabilities, address the demands for high throughput and energy efficiency in modern computational workloads [60]. SIMD instructions facilitate parallel processing, significantly boosting processing speed and efficiency.

The SupeRFIVe methodology demonstrates RISC-V's role in ensuring the functional verification of superscalar processors, essential for maintaining reliability and performance in high-performance computing environments [34]. Additionally, integrating RRAM devices with CMOS technology, as seen in the XNOR-RRAM design, highlights RISC-V's influence on enabling energy-efficient binary operations crucial for deep neural networks [62]. This capability is advantageous for applications demanding high performance and sustainability.

The Janus architecture, utilizing FPGA technology, further illustrates RISC-V's flexibility in supporting reconfigurable computing, allowing processors to dynamically adapt to various computational tasks [63]. This adaptability is vital for addressing the evolving demands of modern computing environments.

RISC-V significantly influences processor design through its open-source framework, promoting innovation and customization while supporting advanced features that enhance performance and energy efficiency. Its modularity enables the development of highly configurable processors, such as dual-issue superscalar designs, achieving improvements in instruction throughput and energy efficiency, particularly in machine learning applications. Furthermore, RISC-V facilitates sophisticated verification methodologies, ensuring functional correctness while balancing performance with sustainability [48, 64, 37, 60, 34]. These attributes position RISC-V as a leading architecture in modern processor development, driving advancements in computational efficiency and adaptability across various domains.

6.2 Customization and Performance Enhancements

RISC-V's open-source architecture offers unparalleled customization opportunities, enabling developers to tailor processor designs to specific applications and performance requirements. This flexibility allows for the creation of specialized extensions and features that enhance computational efficiency across diverse computing environments. RISC-V's modular framework facilitates the integration of custom instructions optimized for specific workloads, particularly in computationally intensive fields such as machine learning, data analytics, and high-performance computing. This adaptability allows for the development of specialized vector instruction sets that significantly improve instruction throughput and energy efficiency [37, 60, 35, 65, 34].

RISC-V enhances performance through custom instruction sets tailored to domain-specific tasks, optimizing the instruction pipeline and reducing execution time. For instance, SIMD extensions in RISC-V processors enable parallel data processing, significantly boosting performance in high-throughput applications [60].

Moreover, RISC-V's support for dynamic execution and reconfigurable computing further enhances its performance potential. Leveraging Field-Programmable Gate Arrays (FPGAs) and other reconfigurable technologies, RISC-V processors can adapt to changing computational demands, optimizing resource allocation and execution efficiency [63]. This adaptability is crucial for applications requiring real-time processing and rapid response to varying workloads.

The integration of advanced memory technologies, such as Resistive Random-Access Memory (RRAM), with RISC-V processors exemplifies the architecture's potential for performance enhancements. The XNOR-RRAM design achieves high throughput and energy efficiency by enabling in-memory computing for binary operations, beneficial for deep learning applications [62]. This integration minimizes data movement between memory and processing units, reducing latency and power consumption.

Furthermore, the SupeRFIVe methodology underscores the importance of comprehensive functional verification in ensuring the reliability and performance of RISC-V-based superscalar processors [34]. By providing a robust framework for verifying complex hardware designs, RISC-V facilitates the development of high-performance processors that meet modern computing environments' rigorous demands.

The customization and performance enhancements offered by RISC-V highlight its transformative impact on processor design. By enabling customized solutions that enhance computational efficiency and adaptability, RISC-V is at the forefront of innovation in high-performance computing (HPC). Its open-source instruction set architecture supports the development of highly configurable processors, such as the dual-issue superscalar RISC-V design, which utilizes advanced techniques like dynamic execution and vector instruction sets to significantly improve performance in diverse applications, including machine learning. This adaptability maximizes instruction throughput and prediction accuracy while addressing the growing demands of various fields, ensuring that RISC-V processors remain integral to the evolution of HPC systems and their applications in cutting-edge research and industry [36, 60, 37].

7 Event-Driven Processing

7.1 Energy Efficiency in Event-Driven Systems

Event-driven processing systems enhance energy efficiency by dynamically responding to real-time events, optimizing resource utilization, and reducing idle times. In high-energy physics, such architectures efficiently manage vast datasets with minimal overhead, improving performance and energy efficiency, as demonstrated in Fermilab's data reconstruction efforts [66, 67, 42, 68, 69]. This model reduces unnecessary power consumption, benefiting applications requiring real-time responsiveness. Efficient communication protocols, like TCP/IP sockets, facilitate rapid data transfer, eliminating delays associated with traditional disk I/O and ensuring timely data processing [66]. By activating processing resources only when necessary, event-driven systems maintain high performance while minimizing energy consumption.

The adaptability of event-driven systems to varying workloads further enhances their energy efficiency. By dynamically reallocating resources in response to real-time demands, these systems optimize energy use and minimize waste, crucial in complex power systems where high-performance and parallel computing are essential for effective grid management and IoT integration [3, 66, 67, 37]. Such adaptability is vital in fields like high-energy physics and spiking neural networks (SNNs), where minimizing the impact of time on computations is essential.

Event-driven processing provides a robust framework for achieving energy efficiency in modern computing environments. By prioritizing event-triggered operations and optimizing data communication, these systems effectively manage complex computational tasks. This is particularly relevant in high-energy physics, where managing large datasets—such as the 50 terabytes produced by Fermilab E791—requires substantial computational resources. Event-driven architectures, like those in the DAMNED framework for SNNs, utilize distributed processing and multithreading to reduce idle time and enhance performance, achieving significant reductions in energy consumption while handling complex simulations [66, 58].

7.2 Applications in High-Energy Physics and SNNs

Event-driven processing is pivotal in high-energy physics and spiking neural networks (SNNs), leveraging its dynamic response capabilities. In high-energy physics, it efficiently handles data from particle collisions, where event timing and sequences are unpredictable, enabling real-time data analysis essential for experiments requiring immediate feedback [13]. The flexibility of event-driven architectures enhances computational efficiency by selectively activating processing resources based on specific events, optimizing power efficiency and performance, and managing peak data loads while minimizing energy consumption. Resource-aware architectures, such as invasive tightly coupled processor arrays (TCPAs), allow applications to claim and release resources according to real-time demands, addressing challenges related to power consumption and reliability in high-performance computing environments [66, 38, 42, 69, 61].

In SNNs, event-driven processing aligns with biological neural coding mechanisms, allowing for high temporal precision in simulating neural activity. This model efficiently handles sparse, asynchronous data, reducing computational load and power consumption compared to traditional neural networks [13]. Applications of event-driven processing in SNNs span robotics, sensory processing, and real-time decision-making systems. By mimicking the dynamics of biological neural networks, SNNs achieve notable energy efficiency and performance, ideal for applications requiring rapid and adaptive responses to changing conditions. Frameworks like DAMNED enhance this efficiency through distributed and multithreaded simulation techniques, optimizing sparse spike emissions typical of SNNs. Innovations such as the Parallel Spiking Unit (PSU) further decouple leaky integration and firing mechanisms, allowing concurrent processing of multiple neuron states, accelerating computation and improving simulation speed while increasing energy efficiency [70, 58].

The integration of event-driven processing in high-energy physics and SNNs showcases its potential to improve computational efficiency and responsiveness. Frameworks like DAMNED enable effective management of large-scale neural network simulations, while advancements in multiprocessor management systems optimize event handling for extensive HEP datasets, such as the 50 terabytes from Fermilab's E791 experiment. Moreover, stream processing frameworks offer scalable performance for continuous high-volume data processing, and process-oriented programming provides an accessible approach to parallel computing, enhancing data-intensive applications. Collectively, these developments underscore the transformative impact of event-driven processing across various scientific and technological domains [66, 67, 58, 68]. By aligning with the dynamic nature of these applications, event-driven systems present a robust framework for advancing research and innovation in areas demanding high precision and adaptability.

8 Processor Design and Optimization

8.1 Design and Optimization Techniques

Processor design and optimization are pivotal for enhancing computational performance, energy efficiency, and adaptability. The Automated Verification of Asynchronous Circuits (AVAC) employs ACL2 to validate asynchronous delay-insensitive primitives, ensuring reliability and efficiency in processor design [9]. Klemen et al. introduce a framework that uses input data size parameters to analyze resource usage, optimizing computational resource allocation and improving system performance [49]. Malafeyev et al. apply congestion game theory to parallel computing, optimizing resource allocation and enhancing efficiency [20].

The DAG-OT method improves schedulability through cache-aware scheduling, enhancing processor performance by optimizing cache utilization and reducing latency in parallel tasks [18]. A source-to-source translator for GAP8 interprets OpenMP directives to generate parallel microcontroller code, optimizing performance in energy-efficient IoT devices [51]. The Wukong framework exemplifies decentralized scheduling for burst-parallel tasks in serverless environments, improving resource utilization and reducing overhead [22].

In asynchronous circuits, the shield model enhances security by monitoring request and acknowledgment signals, identifying potential disruptions [14]. Nishimura et al. classify sequential circuits based on causal functions, crucial for modeling circuit behavior over time [11]. The DAMNED framework utilizes distributed neural network mapping and multithreading, optimizing simulation performance through parallel processing [58]. The FMC method enables real-time processing of analog data with low power consumption, highlighting innovative design techniques for efficient data handling [7].

8.2 Integration of Advanced Computing Concepts

Integrating advanced computing concepts into processor design enhances performance, efficiency, and adaptability. Formal verification techniques are crucial for reliable asynchronous circuit designs, enabling processors to operate efficiently without global synchronization [9]. Understanding stability in asynchronous systems is essential for optimizing processor behavior and integrating asynchronous principles [10]. Game theory in parallel computing offers insights into optimal resource allocation, facilitating efficient computational resource management [20].

The convergence of High-Performance Computing (HPC) and High-Throughput Computing (HTC) demonstrates the potential of advanced concepts to enhance computational efficiency and resource utilization [71]. Expanding OpenMP directives and achieving compilation-level translation are vital for enhancing parallel computing frameworks, as seen in GAP8 translators [51]. The Wukong framework optimizes task scheduling in serverless computing, achieving performance improvements and cost reductions [22]. The shield model's effectiveness in asynchronous circuits highlights the importance of real-time detection of physical attacks, enhancing processor security [14].

The DAMNED simulator combines distributed computing and multithreading to efficiently simulate large-scale Spiking Neural Networks (SNNs), enhancing processing speed and minimizing synchronization issues [58]. Integrating these advanced computing concepts is crucial for achieving enhanced performance and efficiency, paving the way for more resilient and scalable computing systems.

9 Conclusion

The integration of advanced computing concepts into modern processor design has been thoroughly explored in this survey, demonstrating significant improvements in both computational efficiency and system adaptability. Asynchronous circuits offer a compelling alternative to traditional synchronous designs, providing notable enhancements in power efficiency and performance, crucial for applications with stringent power constraints. Superscalar processors effectively leverage instruction-level parallelism to maximize throughput and optimize resource utilization, while parallel computing frameworks facilitate the efficient management of extensive computational tasks. The C-slow technique emerges as a promising method to enhance the performance of embedded systems by enabling concurrent thread execution without the need for additional external resources.

The RISC-V architecture has introduced a paradigm shift in processor design by fostering customization and innovation, allowing for the integration of specialized features tailored to specific computational requirements. Event-driven processing has been pivotal in augmenting computational efficiency and responsiveness, especially in fields such as high-energy physics and spiking neural networks, where real-time data processing is essential.

Future advancements in processor design should focus on refining theoretical models for asynchronous circuits to better address practical constraints and developing robust simulation models for real-world physical processes. Sustainable design approaches, like FSC++, offer promising pathways to reduce the environmental impact of processor designs while maintaining high performance. Additionally, optimizing bottlenecks in OpenMP applications through targeted techniques can lead to considerable performance improvements.

Furthermore, establishing a comprehensive mathematical framework for asynchronous automata is crucial for advancing the modeling and application of these systems. A well-structured curriculum in parallel and distributed computing is essential to enhance understanding and skills among students and professionals, ultimately leading to better outcomes across various disciplines. Benchmarking different frameworks is vital for gaining insights into programming productivity, performance, and energy efficiency, highlighting the importance of context in selecting the appropriate framework.

References

[1] P Balasubramanian. Critique of "asynchronous logic implementation based on factorized dims", 2018.

[2] Jun Ji, Zichen Xi, Bernadeta R. Srijanto, Ivan I. Kravchenko, Ming Jin, Wenjie Xiong, and Linbo Shao. Frequency-domain parallel computing using single on-chip nonlinear acoustic-wave device, 2024.

[3] Ahmed Al-Shafei, Hamidreza Zareipour, and Yankai Cao. A review of high-performance computing and parallel techniques applied to power systems optimization, 2022.

[4] Muhammad Adeel Akram, Aamir Khan, and Muhammad Masood Sarfaraz. C-slow technique vs multiprocessor in designing low area customized instruction set processor for embedded applications, 2012.

[5] Jintao Sun, Zeke Wang, Tao Lu, and Wenzhi Chen. Pyhgl: A python-based hardware generation language framework, 2023.

[6] Yao Xu and Gene Cooperman. Enabling practical transparent checkpointing for mpi: A topological sort approach, 2024.

[7] Cong Wang, Shi-Jun Liang, Chen-Yu Wang, Zai-Zheng Yang, Yingmeng Ge, Chen Pan, Xi Shen, Wei Wei, Yichen Zhao, Zaichen Zhang, Bin Cheng, Chuan Zhang, and Feng Miao. Scalable massively parallel computing using continuous-time data representation in nanoscale crossbar array, 2021.

[8] Rolf Hoffmann. Global cellular automata gca – a massively parallel computing model, 2022.

[9] Freek Verbeek and Julien Schmaltz. Verification of building blocks for asynchronous circuits, 2013.

[10] Serban E. Vlad. Some first thoughts on the stability of the asynchronous systems, 2004.

[11] Shunji Nishimura. Classification of sequential circuits as causal functions, 2023.

[12] Kleovoulos Kalaitzidis. *Advanced speculation to increase the performance of superscalar processors*. PhD thesis, Université Rennes 1, 2020.

[13] Philippe Matherat and Marc-Thierry Jaekel. Concurrent computing machines and physical space-time, 2001.

[14] Radu Mateescu, Wendelin Serwe, Aymane Bouzafour, and Marc Renaudin. Modeling an asynchronous circuit dedicated to the protection against physical attacks, 2020.

[15] Serban E. Vlad. Selected topics in asynchronous automata, 2001.

[16] Peter Tröger. The multi-core era - trends and challenges, 2008.

[17] Subbarao Palacharla, Norman P Jouppi, and JE Smith. Retrospective: Complexity-effective superscalar processors.

[18] Corey Tessler, Venkata P. Modekurthy, Nathan Fisher, and Abusayeed Saifullah. Bringing inter-thread cache benefits to federated scheduling – extended results technical report, 2020.

[19] Hervé Paulino and Nuno Delgado. Cache-conscious run-time decomposition of data parallel computations, 2015.

[20] O. A. Malafeyev and S. A. Nemnyugin. Parallel computing as a congestion game, 2018.

[21] Eric C. Ni, Dragos F. Ciocan, Shane G. Henderson, and Susan R. Hunter. Efficient ranking and selection in parallel computing environments, 2015.

[22] Benjamin Carver, Jingyuan Zhang, Ao Wang, Ali Anwar, Panruo Wu, and Yue Cheng. Wukong: A scalable and locality-enhanced framework for serverless parallel computing, 2020.

[23] Serban E. Vlad. Universal regular autonomous asynchronous systems: omega-limit sets, invariance and basins of attraction, 2010.

[24] Serban E. Vlad. Some properties of the regular asynchronous systems, 2008.

[25] Pál András Papp, Georg Anegg, Aikaterini Karanasiou, and A. N. Yzelman. Efficient multiprocessor scheduling in increasingly realistic models, 2024.

[26] Basel Halak and Hsien-Chih Chiu. Modified micropipline architecture for synthesizable asynchronous fir filter design, 2016.

[27] Weijie Fang, Yanggeng Fu, Jiaquan Gao, Longkun Guo, Gregory Gutin, and Xiaoyan Zhang. Acceleration for timing-aware gate-level logic simulation with one-pass gpu parallelism, 2023.

[28] Robert Clucas, Philip Blakely, and Nikolaos Nikiforakis. Ripple : Simplified large-scale computation on heterogeneous architectures with polymorphic data layout, 2021.

[29] Saber Moradi, Ning Qiao, Fabio Stefanini, and Giacomo Indiveri. A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (dynaps). *IEEE transactions on biomedical circuits and systems*, 12(1):106–122, 2017.

[30] Shubham Yadav. An asynchronous approach for designing robust low power circuits. Master's thesis, University of Twente, 2019.

[31] Daniel Jiménez-González, Carlos Álvarez, Antonio Filgueras, Xavier Martorell, Jan Langer, Juanjo Noguera, and Kees Vissers. Coarse-grain performance estimator for heterogeneous parallel computing architectures like zynq all-programmable soc, 2015.

[32] Shane Carroll and Wei-Ming Ling. A queuing model for cpu functional unit and issue queue configuration, 2018.

[33] Andrea Mondelli. *Revisiting wide superscalar microarchitecture*. PhD thesis, Université de Rennes, 2017.

[34] Andrea Galimberti, Marco Vitali, Sebastiano Vittoria, and Davide Zoni. Functional iss-driven verification of superscalar risc-v processors. In *2024 31st IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 1–4. IEEE, 2024.

[35] Sam Van den Steen and Lieven Eeckhout. Modeling superscalar processor memory-level parallelism. *IEEE Computer Architecture Letters*, 17(1):9–12, 2017.

[36] Xinyao Yi. A study of performance programming of cpu, gpu accelerated computers and simd architecture, 2024.

[37] Claude Tadonki. Hpc curriculum and associated ressources in the academic context, 2018.

[38] Guido Schryen. Speedup and efficiency of computational parallelization: A unifying approach and asymptotic analysis, 2023.

[39] Henry Wong, Vaughn Betz, and Jonathan Rose. High-performance instruction scheduling circuits for superscalar out-of-order soft processors. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, 11(1):1–22, 2018.

[40] Yang Liu, Li Shi, Junwei Zhang, and Thomas G. Robertazzi. Layer based partition for matrix multiplication on heterogeneous processor platforms, 2018.

[41] Kwan-Ho Lee and Chi-yong Kim. A design of a high performance stream processor without superscalar architecture. *Journal of IKEEE*, 21(1):77–80, 2017.

[42] Vahid Lari, Alexandru Tanase, Frank Hannig, and Jürgen Teich. Massively parallel processor architectures for resource-aware computing, 2014.

[43] Erdal Mutlu, Ruiqin Tian, Bin Ren, Sriram Krishnamoorthy, Roberto Gioiosa, Jacques Pienaar, and Gokcen Kestor. Comet: A domain-specific compilation of high-performance computational chemistry. In *International Workshop on Languages and Compilers for Parallel Computing*, pages 87–103. Springer, 2020.

[44] Jayshree Ghorpade, Jitendra Parande, Madhura Kulkarni, and Amit Bawaskar. Gpgpu processing in cuda architecture, 2012.

[45] Yash Shah, Naman Jain, and Utkarsh Gupta. Spsim: Superscalar processor simulater cs305 project report. 2018.

[46] Menglu Yu, Ye Tian, Bo Ji, Chuan Wu, Hridesh Rajan, and Jia Liu. Gadget: Online resource optimization for scheduling ring-all-reduce learning jobs, 2022.

[47] Suejb Memeti, Sabri Pllana, Alecio Binotto, Joanna Kolodziej, and Ivona Brandic. Using meta-heuristics and machine learning for software optimization of parallel computing systems: A systematic literature review, 2018.

[48] Saeideh Sheikhpour, David Metz, Erling Jellum, Magnus Själander, and Lieven Eeckhout. Sustainable high-performance instruction selection for superscalar processors. 2024.

[49] Maximiliano Klemen, Pedro Lopez-Garcia, John P. Gallagher, Jose F. Morales, and Manuel V. Hermenegildo. Towards a general framework for static cost analysis of parallel logic programs, 2019.

[50] Alex V Berka. Space and the synchronic a-ram, 2010.

[51] Reinaldo Agostinho de Souza Filho, Diego V. Cirilo do Nascimento, and Samuel Xavier de Souza. An openmp translator for the gap8 mpsoc, 2020.

[52] Fabian Frei and Koichi Wada. Efficient circuit simulation in mapreduce, 2019.

[53] Brijender Kahanwal. Towards high performance computing (hpc) through parallel programming paradigms and their principles, 2014.

[54] Sascha Hunold and Jesper Larsson Träff. On the state and importance of reproducible experimental research in parallel computing, 2013.

[55] Ardhendu Mandal and Subhas Chandra Pal. An empirical study and analysis of the dynamic load balancing techniques used in parallel computing systems, 2011.

[56] Elankovan Sundararajan and Aaron Harwood. Towards parallel computing on the internet: Applications, architectures, models and programming tools, 2006.

[57] Vibha Rajput and Alok Katiyar. Proactive bottleneck performance analysis in parallel computing using openmp, 2013.

[58] Anthony Mouraud, Didier Puzenat, and Hélène Paugam-Moisy. Damned: A distributed and multithreaded neural event-driven simulation framework, 2006.

[59] János Végh. How deep the machine learning can be, 2020.

[60] Superscalar risc-v processor.

[61] Jürgen Teich, Wolfgang Schröder-Preikschat, and Andreas Herkersdorf. Invasive computing - common terms and granularity of invasion, 2013.

[62] Shihui Yin, Xiaoyu Sun, Shimeng Yu, and Jae sun Seo. High-throughput in-memory computing for binary deep neural networks with monolithically integrated rram and 90nm cmos, 2019.

[63] Janus Collaboration, M. Baity-Jesi, R. A. Banos, A. Cruz, L. A. Fernandez, J. M. Gil-Narvion, A. Gordillo-Guerrero, M. Guidetti, D. Iniguez, A. Maiorano, F. Mantovani, E. Marinari, V. Martin-Mayor, J. Monforte-Garcia, A. Munoz Sudupe, D. Navarro, G. Parisi, M. Pivanti, S. Perez-Gaviro, F. Ricci-Tersenghi, J. J. Ruiz-Lorenzo, S. F. Schifano, B. Seoane, A. Tarancon, P. Tellez, R. Tripiccione, and D. Yllanes. Reconfigurable computing for monte carlo simulations: results and prospects of the janus project, 2012.

[64] Suejb Memeti, Lu Li, Sabri Pllana, Joanna Kolodziej, and Christoph Kessler. Benchmarking opencl, openacc, openmp, and cuda: programming productivity, performance, and energy consumption, 2017.

[65] Cristian Ramon-Cortes, Ramon Amela, Jorge Ejarque, Philippe Clauss, and Rosa M. Badia. Autoparallel: A python module for automatic parallelization and distributed execution of affine loop nests, 2018.

[66] Don Summers, Steve Bracker, Krishnaswamy Gounder, and Kevin Hendrix. An efficient multiprocessor management system for event-parallel computing, 2000.

[67] Adriano Vogel, Sören Henning, Esteban Perez-Wohlfeil, Otmar Ertl, and Rick Rabiser. High-level stream processing: A complementary analysis of fault recovery, 2024.

[68] Edward Givelberg. Process-oriented parallel programming with an application to data-intensive computing, 2014.

[69] Ioannis Chatzigiannakis, Georgios Giannoulis, and Paul G. Spirakis. Energy and time efficient scheduling of tasks with dependencies on asymmetric multiprocessors, 2008.

[70] Yang Li, Yinqian Sun, Xiang He, Yiting Dong, Dongcheng Zhao, and Yi Zeng. Parallel spiking unit for efficient training of spiking neural networks, 2024.

[71] Rudi Killian. *Dynamic superscalar grid for technical debt reduction*. PhD thesis, Cape Peninsula University of Technology, 2018.