Computer Science Review 59 (2026) 100853

Contents lists available at ScienceDirect

Computer Science Review

journal homepage: www.elsevier.com/locate/cosrev

Intra-node transaction parallelism in blockchains: Models, solutions, and trends

Bin Yu [a], Tong Zhou [b,*], He Zhao [b,*], Xiaoyan Li [a], Yuhui Fan [a], Lei Chen [a]

[a] Huainan Normal University, 232038 Huainan, China
[b] Hefei Institutes of Physical Science, Chinese Academy of Sciences, 230031 Hefei, China

ARTICLE INFO

Keywords:
Blockchain
Scalability
Intra-node parallelism
Deterministic parallelism
Optimistic parallelism
Transaction dependency

ABSTRACT

Blockchain technology has been widely adopted across diverse domains, yet its scalability remains a critical bottleneck. Traditional serial transaction execution limits throughput to a low level, failing to meet the demands of high-frequency trading. While parallelization solutions exist, existing literature predominantly focus on "broad parallelism" like sharding and cross-chain. But they overlook "intra-node parallelism", which is a lightweight approach to optimize transaction execution within single nodes without altering core protocols. We aim to fill this gap by conducting a focused, systematic analysis of intra-node transaction parallelism in blockchains. The research methods include: (1) Categorizing intra-node parallelism into three core models (deterministic, optimistic, emerging) based on conflict-handling mechanisms and architectural paradigms. (2) Analyzing ~20 representative solutions to evaluate their core mechanisms, performance trade-offs, and applicable scenarios. (3) Investigating critical practical considerations, including conflict density-based applicability, overhead trade-offs, and synergy with consensus/network layers. (4) Comparing models across dimensions (conflict handling, performance, complexity) to identify strengths and limitations. Key results show that: (1) Deterministic models achieve ~2–3x serial throughput with negligible rollbacks, making them ideal for high-conflict environments. (2) Optimistic models reach ~5–10x serial throughput in low-conflict scenarios but suffer from rollback overhead in high-conflict settings. (3) Emerging models offer breakthrough scalability but require ecosystem changes. (4) No single model dominates. Optimal selection depends on conflict density, contract complexity, and compatibility needs. This study provides a foundational framework for researchers to navigate intra-node parallelism and for practitioners to select or design solutions balancing scalability, consistency, and decentralization. It advances blockchain scalability research by highlighting lightweight, backward-compatible optimizations that complement broad parallelism, enabling the development of high-performance blockchain systems for real-world applications.

1. Introduction

Blockchain technology [1] has been widely adopted across diverse domains such as government services [2], healthcare [3], and supply chain [4], leveraging its inherent advantages of decentralization, immutability, and transparency. However, the scalability of traditional blockchain systems remains a critical bottleneck hindering their further expansion into large-scale applications [5–8]. For instance, Bitcoin's transaction processing capacity is limited to around 7 transactions per second (TPS), and Ethereum, despite its more advanced architecture, struggles with significant congestion during peak times, processing only up to 15 TPS under ideal conditions. In contrast, real-world applications such as high-frequency financial trading require thousands of TPS to function efficiently, and large-scale e-commerce platforms during peak sales periods may need even higher throughput to handle the surge in transactions. This highlights the urgent need to enhance blockchain performance to meet the demands of such high-performance scenarios.

A key limitation of traditional blockchain systems lies in the serial execution of transactions based on a predefined order, which severely restricts throughput and increases latency. This mode of operation is akin to a single-lane road where vehicles (transactions) must follow one another in a strict sequence, leading to significant delays during peak traffic. In contrast, a multi-lane highway allows multiple vehicles to travel simultaneously, significantly improving overall traffic flow.

* Corresponding authors.
E-mail addresses: tzhou@hfcas.ac.cn (T. Zhou), zhaoh@hfcas.ac.cn (H. Zhao).

https://doi.org/10.1016/j.cosrev.2025.100853
Received 18 August 2025; Received in revised form 13 October 2025; Accepted 4 November 2025
Available online 8 November 2025

Similarly, Transaction parallelism [9], which enables simultaneous execution of multiple transactions, has emerged as an effective approach to address blockchain scalability challenges. Existing parallelization solutions can be broadly categorized into four types: inter-network parallelism, inter-subnetwork parallelism, inter-role parallelism, and intra-node parallelism, and they are described in detail in Section 4. Among these, inter-network, inter-subnetwork, and inter-role parallelism typically require fundamental modifications to blockchain architectures or network structures, falling under the scope of "broad parallel execution". In contrast, intra-node parallelism focuses on optimizing transaction execution within individual nodes without altering the core blockchain architecture, making it a pragmatic and lightweight solution for performance enhancement. Jiang et al. [10] highlight that energy blockchains require 500–1000 TPS to handle real-time meter data and transaction settlements. Their survey identifies "execution layer optimization" as a critical unmet need. Similarly, Luo et al. [11] note that 6G-enabled wireless blockchains face latency constraints that broad parallelism cannot fully resolve due to cross-shard communication delays. Intra-node parallelism emerges as a key enabler for 6G-blockchain integration, aligning with the need for "trustworthy, low-latency connectivity" outlined in their work.

Despite the growing body of research on blockchain parallelization, existing survey papers suffer from three critical limitations that hinder their practical and academic value, particularly for intra-node parallelism: (1) Overemphasis on broad parallelism while neglecting node-level optimizations. Some surveys focus heavily on sharding, cross-chain interoperability, or inter-role task allocation, but dedicate minimal attention to intra-node parallelism. It is an oversight given that intra-node optimizations can be deployed without overhauling existing blockchain ecosystems. This gap leaves researchers and practitioners without guidance on lightweight, backward-compatible scalability solutions. (2) Insufficient depth in model-solution mapping and performance trade-offs. Existing reviews either lack quantitative analysis of intra-node solutions or fail to link theoretical parallel models to practical implementations. This ambiguity makes it difficult for practitioners to select solutions aligned with their use case's conflict density or contract complexity. (3) Under-exploration of node-specific challenges and cross-layer synergy. Some surveys discuss concurrency trade-offs but overlook intra-node-specific issues like thread synchronization overhead, hardware-aware scheduling, or lightweight conflict detection. Additionally, no existing work systematically explores how intra-node parallelism synergizes with other layers.

This paper aims to fill these gaps by conducting a focused, systematic review of intra-node transaction parallelism. Unlike existing surveys, a core novelty lies in our parallelism classification framework (deterministic, optimistic, emerging parallelism). It is a structured taxonomy that advances beyond earlier works by explicitly linking parallelism design to conflict-handling mechanisms and architectural intent. Existing surveys either use overly broad categorizations or ignore intra-node models entirely. Our framework resolves this ambiguity, providing a unified lens to compare solutions and their practical trade-offs. This work is designed to directly overcome the limitations above: (1) To address the neglect of intra-node parallelism, we center the entire review on node-level execution, providing a detailed taxonomy and analysis of solutions that require no core architectural changes. (2) To resolve ambiguous model-solution mapping, we link 20+ representative solutions to their underlying parallel models and quantify trade-offs. (3) To explore node-specific challenges and cross-layer synergy, we dedicate Section 7 to analyzing intra-node overheads and how intra-node parallelism complements consensus and network optimizations.

The main contributions are summarized as follows: (1) Systematic classification of intra-node parallelism models. This study establishes a clear taxonomy by categorizing intra-node parallelism into three core models based on conflict-handling mechanisms and architectural design. This classification addresses the lack of standardized categorization in existing literature. (2) Comprehensive analysis of representative solutions. We conduct an in-depth evaluation of approximately 20 representative intra-node parallelism solutions, quantifying their performance, core technologies, and applicable scenarios. This analysis fills the gap of insufficient empirical and technical detail in existing surveys, offering actionable insights for solution selection. (3) Holistic exploration of practical trade-offs. The research goes beyond technical mechanisms to investigate critical practical considerations, including how conflict density and contract complexity influence model applicability, the overhead of parallelism, and the synergy between intra-node parallelism and other layers. This holistic view helps practitioners navigate trade-offs between scalability, consistency, and efficiency. (4) Comparative framework and future directions. We develop a cross-model comparison framework to identify scenario-specific optimal models. Additionally, we outline future trends that address current limitations, guiding researchers toward impactful areas of further study.

The subsequent sections of this paper are structured to systematically achieve the research goals and elaborate on the contributions outlined above: Section 2 provides foundational background on transaction execution modes, core concepts, and key parallelism challenges. Section 3 reviews existing blockchain parallelism surveys, highlighting their limitations regarding intra-node parallelism. Section 4 defines the survey scope by distinguishing intra-node parallelism from broad parallel execution and explaining its prioritization, then describes survey methodology. Section 5 details the three core intra-node parallelism models (deterministic, optimistic, emerging) with their principles and strengths/limitations. Section 6 analyzes ~20 representative solutions for each model via comparative tables, covering core technologies, performance, and use cases. Section 7 discusses practical implications including applicability, overhead trade-offs, and cross-layer scalability. Section 8 compares the three models, synthesizes standardized benchmarking, and outlines limitations and future trends in intra-node parallelism. Section 9 synthesizes key findings and restates the paper's contributions.

2. Background

2.1. Transaction execution

In blockchain systems, transaction execution is a core process that transforms the global state of the network through predefined rules encoded in smart contracts [12,13] or protocol logic. A transaction [14] typically contains some basic information, such as sender address, recipient address, value transferred, or function calls to smart contracts, etc. It also includes cryptographic signatures for authentication and other auxiliary data like gas limits in Ethereum [15].

Traditional blockchain systems, such as Bitcoin [1] and early versions of Ethereum, adopt a serial execution mode for transactions within a block. In this mode, transactions are processed one after another in a strictly predefined order, determined by the block proposer or consensus algorithm. The serial execution flow follows three key steps, which is shown in Fig. 1.

1. Transaction Ordering. Transactions in a block are arranged in a linear sequence, often based on their arrival time, gas price, or consensus rules. This sequence is immutable once the block is finalized.
2. Transaction Execution. Each transaction is executed sequentially in the specified order. The execution engine reads the current state of relevant accounts or smart contracts, such as account balances and contract storage variables, etc., processes the transaction logic, and writes new values to update the state.
3. State Updating. After a transaction is executed successfully, its state changes are committed to the global state database before the next transaction begins. If a transaction fails (e.g., due to insufficient

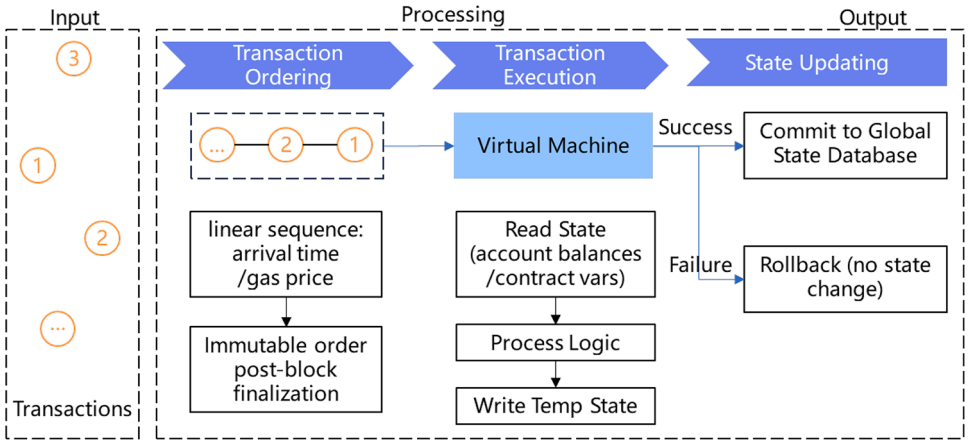

Fig. 1. Transactions serial execution.

funds or invalid logic, etc.), it is rolled back, and the state remains unchanged, ensuring no impact on subsequent transactions.

The serial execution mode guarantees determinism and consistency. All nodes executing the same set of transactions in the same order will reach an identical final state. This simplicity is critical for maintaining blockchain's decentralized trust, as it eliminates ambiguities in state synchronization across nodes. However, this mode suffers from severe performance limitations. It cannot effectively leverage multi-core hardware architectures, and even independent transactions with no overlapping state access must wait for prior transactions to complete. As a result, throughput (measured in transactions per second, TPS) is constrained by the execution speed of a single thread, making serial execution a major bottleneck for blockchain scalability in high-demand scenarios.

2.2. Parallel execution

To address the scalability limitations of serial execution, parallel transaction execution has emerged as a key optimization strategy. Parallel execution enables multiple transactions to be processed simultaneously across multiple threads or cores within a single node. Unlike serial execution, which enforces a strict linear order, parallel execution leverages the independence between transactions to maximize hardware utilization, thereby increasing throughput and reducing latency. The core idea is to identify and execute non-conflicting transactions in parallel while ensuring the final state remains consistent across all nodes. Fig. 2 presents the process of transactions parallel execution.

To understand how parallel execution works, several fundamental concepts must be clarified.

Smart Contract: A self-executing program stored on the blockchain that automates the execution of predefined rules (such as asset transfers, condition-based payments, etc.). Smart contracts manage and modify the blockchain's state through functions, making them central to transaction logic and state interactions.

State: The current snapshot of data maintained by the blockchain, including account balances, contract storage variables, configuration parameters, etc. The state is globally consistent across all nodes in a decentralized network and is updated through transaction execution.

Read and Write Set: The Read Set is a collection of state variables or data items that a transaction reads during execution. The Write Set is a collection of state variables or data items that a transaction modifies, such as the new balance of an account after a transfer, updated contract parameters, etc. These sets help identify dependencies between transactions. If two transactions share overlapping items in their read or write sets, they may conflict.

Transaction Dependency: A relationship between two transactions where the execution result of one affects the other. Dependencies arise when: Transaction *Tx1* reads a state variable modified by Transaction *Tx2* (read-after-write dependency); Transaction *Tx1* modifies a state variable read by Transaction *Tx2* (write-after-read dependency); Both *Tx1* and *Tx2* modify the same state variable (write-after-write

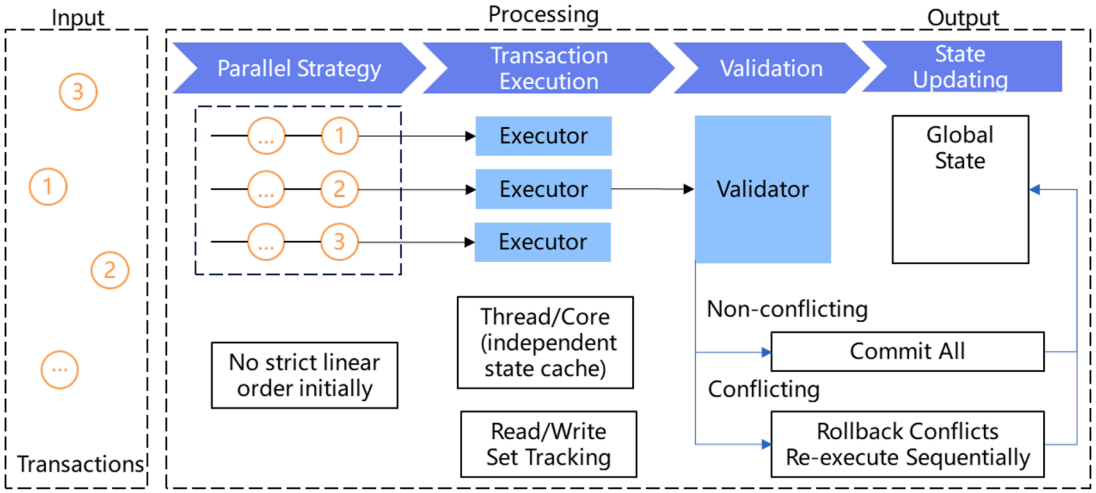

Fig. 2. Transactions parallel execution.

dependency). Transactions with dependencies must be executed in a specific order to avoid state inconsistencies.

Conflict: A situation where parallel execution of two or more transactions would lead to an inconsistent or incorrect final state due to overlapping read/write sets. Conflicts are the primary barrier to parallel execution and must be detected and resolved to ensure correctness.

Rollback Rate: The percentage of transactions that are aborted and re-executed sequentially due to unresolved conflicts during parallel execution. A high rollback rate indicates inefficiencies in conflict detection or scheduling, as resources are wasted on invalid parallel attempts.

In the parallel execution mode, the goal is to minimize conflicts or resolve them efficiently while maximizing the number of transactions processed simultaneously. By analyzing read/write sets, identifying dependencies, and applying conflict resolution strategies, parallel execution engines can leverage multi-core hardware to significantly improve blockchain performance without sacrificing state consistency.

2.3. Challenges and key technologies

2.3.1. Core challenges

Intra-node parallel transaction execution holds promise for enhancing blockchain scalability but confronts unique challenges centered on balancing performance, consistency, and efficiency.

1. State Conflict and Final Consistency. The most critical challenge is ensuring state consistency amid parallel execution. When transactions access overlapping state variables, conflicts arise. Unresolved conflicts can lead to divergent states across nodes, violating the blockchain's fundamental property of decentralized consensus. For example, Aptos' Block-STM [16,17] experiences 8–12 % rollback rates in high-conflict DeFi scenarios. These rollbacks not only undo invalid state changes but also waste 30–40 % of CPU resources spent on initial execution, as reported in Aptos' 2023 testnet data.
2. Overhead from Parallelism. Parallel execution introduces non-trivial additional workload that may offset performance gains. (1) Conflict detection overhead: Analyzing read/write sets to identify dependencies between transactions, which consumes computational resources. Solana's Gulf Stream [18,19] uses GPU acceleration for dependency analysis, but this still adds 0.01 ms of pre-execution latency per transaction. (2) Rollback costs: In optimistic models, transactions executed in parallel may later be found to conflict, requiring rollbacks to undo invalid state changes, wasting resources spent on initial execution. Monad [47] reports that rollbacks consume 20–25 % of total execution time when conflict rates exceed 20 %. In extreme cases (85 % conflict), rollback overhead reduces throughput to 12k TPS, which is below the 15k TPS of some optimized serial execution engines. (3) Synchronization overhead: Coordinating threads/cores to avoid race conditions during state access, which can introduce latency if not optimized. DMVCC [20] requires 2.3x more RAM to maintain thread-local state snapshots. This memory overhead forces some small-scale validators to exit testnets.
3. Parallel Efficiency Limitations. Maximizing parallel efficiency (ratio of parallel throughput to serial throughput × core count) is constrained by real-world transaction dependencies and hardware limits. Solana achieves only ~45 % parallel efficiency with 32 cores (65k TPS actual vs. 144k TPS theoretical) [18,19], as 5–10 % of transactions in typical blocks share state dependencies.

2.3.2. Key technologies

To address the above challenges, several key technologies have been developed (Fig. 3), all of which form the foundation of intra-node parallel execution.

1. Conflict Detection and Dependency Analysis. There are two primary approaches to identify transaction dependencies. (1) Static analysis: Performed before execution, this method parses transaction inputs and smart contract bytecode to precompute read/write sets, building a dependency graph to group independent transactions. Static analysis is lightweight and fast but may miss dynamically determined state accesses or rely on accurate client-provided information.

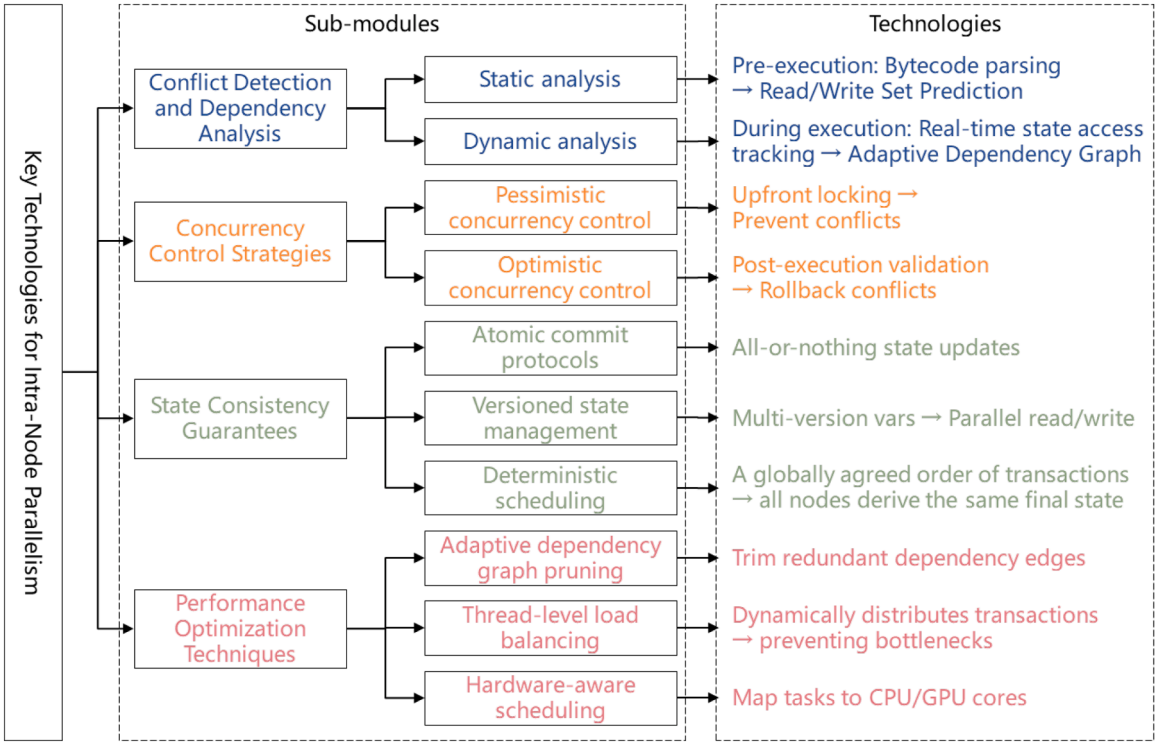

Fig. 3. Key technologies for intra-node parallelism.

(2) Dynamic analysis: Conducted during execution, this tracks real-time state accesses to identify dependencies as they emerge, adapting to runtime changes in transaction behavior. While more accurate, dynamic analysis introduces additional execution overhead.

2. Concurrency Control Strategies. These strategies manage conflicts to ensure consistency. (1) Pessimistic concurrency control: Prevents conflicts upfront by enforcing order constraints. This approach avoids rollbacks but may limit parallelism in high-conflict scenarios. (2) Optimistic concurrency control (OCC): Allows transactions to execute in parallel without prior locking, then validates for conflicts post-execution. Conflicting transactions are rolled back and re-executed sequentially. OCC improves parallelism in low-conflict scenarios but requires efficient rollback mechanisms to handle conflicts detected after execution.
3. State Consistency Guarantees. Mechanisms to ensure final state consistency across parallel threads include: (1) Atomic commit protocols. Ensure that either all state changes from a set of parallel transactions are committed, or none are, preventing partial updates. (2) Versioned state management. Maintains multiple versions of state variables to allow parallel reads/writes without blocking, with conflicts resolved via version comparison during validation. (3) Deterministic scheduling. Enforces a globally agreed order of transactions even when executed in parallel, ensuring all nodes derive the same final state.
4. Performance Optimization Techniques. To enhance parallel efficiency, technologies focus on reducing overhead and maximizing resource use: (1) Adaptive dependency graph pruning. Optimizes dependency graphs by removing redundant edges, reducing analysis time. (2) Thread-level load balancing. Dynamically distributes transactions across cores based on workload, preventing bottlenecks. (3) Hardware-aware scheduling. Leverages multi-core architectures to minimize latency in state sharing between threads, ensuring efficient utilization of available hardware resources.

These challenges and key technologies form the core of intra-node parallel execution research, guiding the design of parallel models and solutions discussed in subsequent sections.

3. Related work

The quest for scalability in blockchain systems has spurred extensive research on parallelization, with numerous survey papers synthesizing progress in this domain. These surveys have laid foundational understanding but often focus on broad parallelism strategies, such as sharding and cross-chain interoperability, while neglecting the nuances of intra-node parallelism. This section reviews key existing parallelism surveys, analyzes their limitations to clarify core gaps, and expands on domain-specific studies to introduce a critical, unaddressed gap in application-aligned parallelism research.

3.1. Analysis of existing parallelism surveys

Table 1 summarizes five surveys in blockchain parallelism, with a refined focus on how their limitations perpetuate gaps in intra-node parallelism research.

3.2. Research gaps

While existing surveys have significantly advanced understanding of blockchain parallelism, critical gaps remain.

1. Narrow focus on node-level parallelism. Most surveys emphasize broad parallelism, such as sharding, cross-chain, or smart contract parallelism in isolation, but lack systematic analysis of intra-node parallelism. This includes thread/cores-level parallel execution within a single node, which is essential for lightweight, backward-compatible optimizations.
2. Insufficient model-solution stratification. Few surveys explicitly distinguish between theoretical parallel models and their practical implementation schemes. This lack of clarity hinders the mapping of theoretical concepts to real-world solutions, making it difficult for practitioners to apply these models effectively.
3. Limited attention to node-specific challenges. Challenges unique to intra-node parallelism, such as thread synchronization overhead, hardware-aware scheduling, and lightweight conflict detection, are under-explored in existing reviews. These challenges are crucial for optimizing performance at the node level without altering the core blockchain architecture.
4. Neglect of parallelism in domain-specific blockchain applications. The AIoT-supply chain study [28] proposes a blockchain framework to process real-time sensor data and optimize green supply chain operations. It emphasizes energy efficiency and low latency but relies on traditional serial transaction execution. Intra-node parallelism like optimistic models for low-conflict sensor data could directly enhance throughput while aligning with the study's "green" goal of optimizing CPU utilization to reduce energy waste. The water IoT study [29] designs a blockchain-digital twin architecture to standardize high-volume water quality monitoring data. It prioritizes resource efficiency for edge devices but ignores that serial execution will limit scalability for sensor streams. Lightweight intra-node techniques of adaptive parallelism granularity for edge nodes could solve this. This gap creates a disconnect between parallelism research and real-world applications, where blockchain is increasingly adopted but held back by serial execution bottlenecks.

Table 1
Blockchain Parallelism review papers.

Review	Year	Focus	Contributions	Strengths	Limitations
[21]	2019	early smart contract parallelization	pioneered the sharding-parallelism-consensus framework; classified static/dynamic parallelization	established foundational theoretical boundaries; focused on challenges of serial execution	outdated coverage, lacking modern architectures like DAG/Block-STM; limited to smart contracts
[22]	2022	empirical comparison of parallel execution schemes	quantified performance metrics (TPS, latency, conflict rate) for Ethereum, Hyperledger Fabric, Aptos, etc.	solid experimental data from 100-node clusters; practical guidance for technology selection	limited to financial/asset contracts; omitted emerging architectures
[23]	2024	broad parallelization across the entire blockchain architecture	distinguished intra-chain vs. inter-chain parallelism; classified by granularity (instruction/ transaction/block/chain)	holistic perspective beyond smart contracts; aligned technologies with real-world scenarios	insufficient depth in core mechanisms; lack of performance boundary analysis
[24]	2021	parallelism for smart contracts with a focus on Chinese blockchains	categorized models into static/dynamic/inter-node/divide-and-conquer parallelism; analyzed FISCO BCOS [25]/ AntChain [26] cases	practical classification aligned with engineering needs; strong focus on domestic industrial solutions	limited coverage of international frontiers; weak theoretical depth in security-performance trade-offs
[27]	2025	systematic synthesis of concurrency with cognitive correction	proposed a concurrency trilemma (security/ performance/compatibility)	cutting-edge coverage of 2024 technologies; quantified conflict-entropy metrics	limited engineering case studies; complex theoretical models

This survey directly resolves the four gaps by: (1) Centering the entire analysis on intra-node parallelism, providing a roadmap for lightweight, backward-compatible scalability. (2) Linking 20+ intra-node solutions to their underlying models (deterministic, optimistic, emerging) with quantitative trade-offs like TPS and rollback rates. (3) Dedicating Section 7.2 to node-specific challenges (e.g., synchronization overhead, memory constraints) and mitigation strategies. (4) Connecting intra-node parallelism to domain-specific needs (Section 7.1), filling the gap between parallelism research and application-focused studies like [28] and [29].

4. Survey scope and methodology

To provide a clear and focused analysis of intra-node transaction parallelism, this survey delineates its scope by distinguishing between broad parallel execution and narrow parallel execution. This distinction is crucial for understanding the specific context and applicability of the solutions discussed in this paper. Fig. 4 presents broad parallel execution and narrow parallel execution.

4.1. Broad parallel execution

Broad parallel execution refers to scalability solutions that achieve parallelism through fundamental modifications to the blockchain's architecture, network structure, or consensus mechanism. These approaches typically distribute transaction processing across multiple independent sub-systems, such as chains, shards, or sub-networks, and require coordinated changes to the core protocol. Key categories include:

1. Inter-network parallelism. Parallel execution across distinct, independent blockchain networks, enabled by interoperability protocols that connect separate chains. (1) Cross-chain parallelism: Transactions are processed simultaneously across unrelated blockchains via bridges, such as Polkadot [30], Chainlink [31], Axelar [32], which synchronize state and assets between networks. (2) Sidechain/sub-chain parallelism: Auxiliary chains, such as Bitcoin's Liquid sidechain [33], Ethereum's Layer 2 solutions like Arbitrum [34], operate in parallel with a main chain, handling specific transaction types and periodically settling final states on the main chain. This form of parallelism requires standardized communication protocols, such as Polkadot's parachains [30], Cosmos' Inter-Blockchain Communication [35], and trust mechanisms to validate cross-network interactions.
2. Inter-subnetwork parallelism. Parallel processing within a single blockchain network by partitioning it into independent sub-networks, each responsible for a subset of transactions. (1) Sharding: Networks like Ethereum 2.0 [36] and Zilliqa [37] divide nodes into shards, where each shard processes its assigned transactions in parallel. Cross-shard transactions are routed through a beacon chain to ensure global state consistency. (2) Channel-based parallelism: Permissioned blockchains like Hyperledger Fabric [38] use private channels to isolate transaction flows, allowing separate groups of nodes to process channel-specific transactions in parallel without interfering with the main network. Inter-subnetwork parallelism relies on efficient transaction partitioning to minimize cross-subnetwork dependencies, and scalable cross-subnetwork consensus, often requiring trade-offs between decentralization and performance.
3. Inter-role parallelism. Parallel task allocation among nodes with distinct, specialized roles within a single blockchain network. Instead of requiring every node to perform all functions, including ordering, executing, and validating, etc., the network decomposes its workflow into discrete tasks and assigns them to dedicated node groups. In the Flow [39–41] blockchain, nodes are divided into collectors, consensus nodes, execution nodes, and verification nodes. Each role operates in parallel, with execution nodes focusing exclusively on transaction processing while consensus nodes handle ordering. In Hyperledger Fabric [38], endorsers, orderers, and committers form distinct roles, enabling parallel workflows where endorsement and ordering can proceed concurrently for non-conflicting transactions. This role-separation reduces redundant work across nodes but requires redesigning network protocols to coordinate task handoffs and ensure trust between role-specific nodes.

Broad parallel execution often relies on efficient transaction propagation to feed intra-node execution engines, which is underscored by Lai et al. [42]. While broad parallel execution significantly enhances scalability, it introduces substantial complexity. It demands architectural overhauls, robust synchronization mechanisms across parallel components, and often new trust models to secure distributed processing. These challenges highlight the need for complementary, lightweight optimizations, such as the intra-node parallelism focused on in this

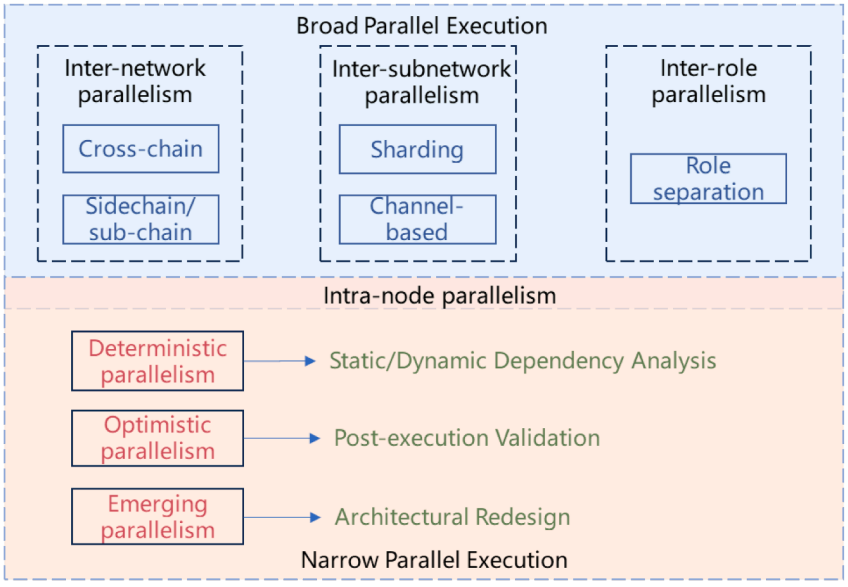

Fig. 4. Broad parallel execution and narrow parallel execution.

survey.

4.2. Narrow parallel execution

Narrow parallel execution refers to parallel transaction processing within a single node, leveraging multi-core or multi-threaded hardware architectures without modifying the blockchain's core protocol, network structure, or consensus mechanism. It targets the execution phase of transactions by exploiting independence between transactions to maximize CPU utilization.

Key characteristics of narrow parallel execution include: (1) Scope: Limited to a single node's execution engine; no changes to the blockchain's network topology, consensus rules, or global state synchronization protocols. (2) Granularity: Focuses on transaction-level parallelism, not block-level or chain-level, where independent transactions within a single block are executed simultaneously across threads/cores. (3) Compatibility: Works with existing blockchain architectures as a performance optimization of the execution layer, preserving consensus and network compatibility.

By focusing on narrow parallel execution, this survey addresses a critical gap in existing literature. While broad parallelism has been extensively reviewed, the node-level optimizations that enable lightweight, backward-compatible scalability have not been systematically synthesized. This scope allows for deep analysis of transaction dependency detection, conflict resolution, and thread scheduling, which directly impact a node's ability to process transactions efficiently.

Narrow parallel execution is prioritized in this survey for three reasons. (1) Practicality: It enables performance gains without disrupting existing blockchain ecosystems, making it easier to adopt in mature networks. (2) Hardware efficiency: Modern multi-core processors are underutilized in serial execution models; narrow parallelism unlocks this latent capacity. (3) Complementarity: It works synergistically with broad parallelism (e.g., sharded networks can further benefit from node-level parallelism within each shard), making it a critical component of end-to-end scalability.

4.3. Survey methodology

To ensure a systematic, rigorous analysis of intra-node transaction parallelism, this survey follows a structured four-step review process to address gaps in transparency about how solutions and insights were curated.

Step 1: Literature and Solution Identification. We first defined search criteria to identify relevant studies and intra-node parallelism solutions.

1. Academic Databases. Searching IEEE Xplore, ACM Digital Library, and Google Scholar (2016–2025) using keywords: intra-node parallelism, blockchain transaction parallel execution, blockchain deterministic parallelism, blockchain optimistic parallelism, and blockchain emerging parallelism.
2. Technical Documentation. Including solutions from leading blockchain projects by reviewing official whitepapers, technical blogs, and GitHub repositories, ensuring coverage of both peer-reviewed and industry-adopted approaches.
3. Inclusion and Exclusion Criteria. Including solutions that (1) focus on transaction execution within a single node (not sharding or cross-chain), (2) provide measurable performance data (e.g., TPS, rollback rates), and (3) have public technical details. Excluding broad parallelism studies and preliminary solutions with no implementation data. This yielded ~50 candidate studies and solutions, which were narrowed to 20 representative ones for in-depth analysis.

Step 2: Categorization into Parallelism Models. The ~20 selected solutions were categorized into three core models (deterministic, optimistic, emerging) using a two-dimensional framework.

1. Conflict-Handling Mechanism. Whether conflicts are prevented upfront (deterministic), resolved post-execution (optimistic), or minimized via architectural redesign (emerging).
2. Architectural Intent. Whether the solution optimizes existing execution workflows (deterministic and optimistic) or redefines state/computation paradigms (emerging). Each solution was cross-validated against this framework.

Step 3: Data Extraction and Analysis. For each solution, we extracted standardized data points to enable fair comparison.

1. Core Mechanisms: Key technologies, such as a dependency graphs for PEEP and Block-STM for Aptos.
2. Performance Metrics: Throughput, rollback rate, latency, and parallel efficiency, sourced from published experiments or project testnets.
3. Practical Attributes: Backward compatibility, hardware requirements, and applicable scenarios. Data was cross-verified across multiple sources to reduce bias.

Step 4: Synthesis of Insights and Trade-Offs. We synthesized extracted data to identify model-specific strengths, limitations, and deployment trade-offs.

1. Cross-Model Comparison: Evaluating how each model performs under varying conflict rates.
2. Practical Implications: Linking technical metrics to real-world feasibility.
3. Gap Identification: Highlighting unaddressed areas to guide future research.

5. Intra-node parallelism models

Intra-node parallelism modes are shown in Fig. 5.

5.1. Deterministic parallelism model

5.1.1. Overview

The deterministic parallelism model is a class of intra-node parallel execution approaches that guarantee consistent and predictable outcomes by explicitly identifying transaction dependencies before execution. Unlike optimistic models that resolve conflicts post-execution, deterministic models prevent conflicts upfront through rigorous pre-execution analysis, ensuring that parallel execution produces the same result as a serial execution of the same transaction set in their globally agreed order. This determinism is critical for maintaining blockchain state consistency across decentralized nodes.

The deterministic parallelism model operates on three foundational principles. (1) Dependency Awareness: All transaction dependencies are identified before any transaction is executed. This eliminates uncertainty about potential conflicts during parallel processing. (2) Conflict Prevention: Transactions with identified dependencies are forced to execute in a predefined order, while independent transactions are grouped for parallel execution. This ensures no conflicting operations occur simultaneously. (3) Outcome Equivalence: The final state after parallel execution must be identical to the state that would result from executing transactions in their original sequential order. This preserves the blockchain's core property of deterministic state transitions.

Existing surveys [21,24] mention "static" or "pessimistic" parallelism but fail to define it as a cohesive model centered on upfront conflict prevention. Our deterministic model advances this by: (1) Establishing three foundational principles (dependency awareness,

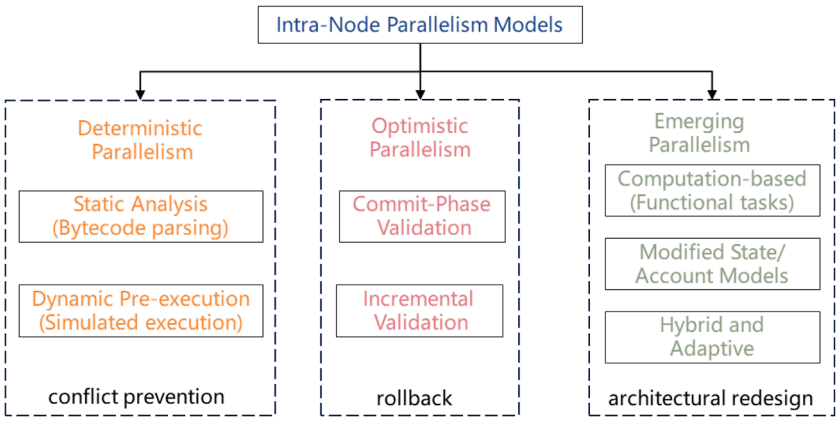

Fig. 5. Intra-node parallelism modes.

conflict prevention, outcome equivalence) that unify disjoint techniques under a single framework. (2) Subclassifying into "static analysis-based" and "dynamic pre-execution" to distinguish between solutions that rely on bytecode parsing and those that use simulated execution.

5.1.2. Classification

Deterministic parallelism models are primarily categorized based on how transaction dependencies are identified, resulting in two main subtypes:

1. Static Analysis-Based

The static analysis-based models derive dependencies by analyzing smart contract code or client-supplied read/write sets without executing the transactions.

(1) Bytecode/Source Code Parsing: Tools parse smart contract bytecode (e.g., Ethereum's EVM bytecode) or high-level source code (e.g., Solidity [43]) to identify potential state variables accessed by each function. For example, a transfer function in a token contract may be statically analyzed to determine it reads and writes the balance variables of two accounts.
(2) Client-Supplied Read/Write Sets: In some blockchain designs, clients explicitly define the read/write sets when submitting transactions. This shifts the burden of dependency identification to the client, reducing the node's computational overhead. Systems like Hyperledger Fabric (in certain configurations) and permissioned blockchains with trusted clients leverage this approach, as it ensures read/write sets are known upfront and eliminates the need for on-node code analysis.

Static models are lightweight and fast, as they avoid the computational cost of transaction execution. However, they have limitations: bytecode parsing may miss dynamically determined state accesses, such as variables accessed via computed addresses, while client-supplied sets rely on trust in clients to avoid misdeclarations, which could lead to hidden conflicts. Both scenarios can result in false dependencies or missed dependencies, either reducing parallelism or risking state inconsistency.

2. Dynamic Pre-Execution

Dynamic pre-execution models identify dependencies by simulating transaction execution in a controlled environment before actual execution. During pre-execution: (1) Transactions are processed in their agreed order, but state changes are not committed to the global state. (2) All read and write operations are recorded to generate precise read/write sets. (3) Dependencies are derived directly from these observed sets, eliminating false positives from static analysis.

For example, a transaction calling a complex DeFi [44] contract with conditional logic (e.g., transfer tokens only if a price threshold is met) will have its actual read/write sets recorded during pre-execution, capturing dynamic state accesses that static analysis might miss. Dynamic models offer higher accuracy but introduce pre-execution overhead, which can impact latency.

5.1.3. Strengths and limitations

The strengths of deterministic parallelism model including: Eliminates rollback overhead (no post-execution conflicts); guarantees state consistency; suitable for high-conflict scenarios, such as DeFi contracts with frequent overlapping state access. There are also some limitations: Pre-analysis introduces latency; static models may overestimate dependencies (reducing parallelism); dynamic models require computational resources for pre-execution. Deterministic parallelism models are widely adopted in permissioned blockchains and enterprise solutions, where consistency and predictability are prioritized over minimal latency. These models are particularly effective in environments where transaction dependencies can be reliably identified upfront, ensuring that parallel execution does not compromise the integrity of the blockchain's state.

5.2. Optimistic parallelism model

5.2.1. Overview

The optimistic parallelism model assumes transactions can be executed in parallel with minimal conflicts, deferring conflict detection and resolution until after execution. Unlike deterministic models that prevent conflicts upfront, optimistic models prioritize maximizing parallelism by allowing simultaneous execution of transactions, only intervening if conflicts are detected during a post-execution validation phase. This approach is optimistic in its assumption that most transactions in a block are independent, making the overhead of potential rollbacks worthwhile for performance gains.

The optimistic parallelism model operates on three key principles. (1) Parallel Execution First. Transactions are executed concurrently across threads/cores without prior dependency checks, leveraging multi-core hardware to maximize throughput. (2) Post-Execution Validation. After parallel execution, a validation phase checks for conflicts between transactions. (3) Selective Rollback. Conflicting transactions are identified and rolled back to their pre-execution state, then re-executed sequentially to ensure state consistency. Non-conflicting transactions are committed to the global state.

This model preserves the blockchain's deterministic outcome by ensuring that the final state matches the result of serial execution in the original transaction order, even if intermediate steps involve parallelism and rollbacks. Prior works [22,27] reference "optimistic execution" but

treat it as a standalone technique rather than a model with consistent design goals. Our optimistic model advances beyond this by: (1) Defining three key principles (parallel execution priority, post-execution validation, selective rollback). (2) Categorizing by validation timing (commit-phase vs. incremental) to explain trade-offs. It is a distinction missing in [27]'s "concurrency trilemma", which ignores how validation timing impacts the security-performance balance.

5.2.2. Classification

Optimistic parallelism models are primarily categorized based on when and how conflicts are detected during the execution lifecycle.

1. Commit-Phase Validation

Transactions are executed in parallel, and conflicts are checked only when transactions are ready to commit to the global state. Transactions read from and write to temporary state spaces to avoid blocking each other. Before finalizing state changes, a global validation step compares read/write sets across all parallel transactions. If conflicts are found, one or more conflicting transactions are rolled back.

The Optimistic Concurrency Control (OCC) mechanism in databases, adapted for blockchains like Aptos. Aptos' Block-STM (Software Transactional Memory) uses commit-phase validation to check for overlapping state accesses in parallel-executed transactions. Commit-phase validation minimizes runtime overhead during execution but may incur larger rollback costs if many conflicts are detected late.

2. Incremental Validation

The incremental validation checks for conflicts during execution, allowing early detection and mitigation of conflicts. As transactions execute, their read/write sets are recorded and checked against those of other parallel transactions in real time. If a conflict is detected mid-execution, The latter transaction is rolled back immediately, avoiding wasted effort on completing an invalid transaction.

Sui's parallel execution engine uses incremental validation with object ownership tracking. If a transaction attempts to modify an object already accessed by another parallel transaction, it is aborted early. Incremental validation reduces the cost of rollbacks by catching conflicts sooner but introduces slight runtime overhead from continuous checks.

5.2.3. Strengths and limitations

The strengths of optimistic parallelism model including: High parallel efficiency in low-conflict scenarios; Low latency for independent transactions; Compatibility with complex contracts, optimistic execution handles dynamically determined state accesses without false dependencies. There are also some limitations: Rollback overhead in high-conflict scenarios; Complex state management; non-deterministic intermediate states.

Optimistic parallelism models are most effective in low-conflict transaction environments and latency-sensitive applications. They are particularly suitable for blockchains with complex smart contracts where the benefits of parallel execution outweigh the costs of occasional rollbacks. Conversely, they are less suitable for high-conflict scenarios where rollback costs can negate the performance gains from parallelism.

5.3. Emerging parallelism model

5.3.1. Overview

As blockchain applications grow in complexity, emerging parallelism model has been developed to address the limitations of traditional deterministic and optimistic approaches. It is defined as intra-node parallelism paradigms that redefine core blockchain architectures (state representation, computation logic, or execution workflows) to eliminate inherent dependencies. Emerging parallelism's core distinction lies in proactive dependency reduction through architectural redesign. It does not optimize how conflicts are "handled" (prevention/rollback) but instead minimizes or eliminates conflicts at the source by restructuring state, computation, or transaction logic.

No prior survey [21–24,27] recognizes emerging parallelism as a separate model. Our emerging model fills this gap by: (1) Focusing on solutions that redefine core blockchain components (state models, computation paradigms) rather than optimizing execution order. (2) Classifying into "computation-based", "modified state/account", and "hybrid adaptive" subcategories to highlight how these solutions address inherent dependencies.

5.3.2. Classification

Below are key types of emerging parallelism models, each tied to architectural reconfiguration that eliminates dependencies.

1. Computation-based parallelism

This model reimagines smart contract execution as a parallelizable computational process, leveraging stateless or functionally pure logic to minimize dependencies. By structuring computations to avoid hidden state interactions, it enables fine-grained parallelism at the instruction or sub-task level.

Computations are decomposed into independent, deterministic sub-tasks with explicit input-output relationships, eliminating implicit state dependencies. This allows parallel execution without prior dependency checks, as conflicts are inherently avoided through design. Some emerging frameworks use machine learning to predict transaction conflict probabilities. Based on predictions, the engine allocates transactions to deterministic or optimistic execution paths, balancing latency and rollback overhead.

2. Modified state/account models

Traditional account-based models, such as Ethereum's balance-centric accounts, centralize state in shared variables, creating inherent dependencies that limit parallelism. Emerging models redesign state representation to isolate data, enabling parallel access by default.

State is partitioned into discrete, independent units like objects, tokens, or UTXOs with clear ownership or isolation rules. Transactions interact with specific units, and parallel execution is permitted for transactions targeting non-overlapping units, as conflicts are structurally prevented.

3. Hybrid and adaptive parallelism

These models combine elements of deterministic and optimistic approaches, dynamically adjusting strategies based on real-time transaction characteristics, such as conflict rates, complexity, to optimize performance.

The model switches between pre-execution dependency analysis (deterministic) and post-execution validation (optimistic) using adaptive heuristics. For example, high-conflict scenarios trigger stricter upfront checks to reduce rollbacks, while low-conflict periods prioritize optimistic execution to maximize throughput.

5.3.3. Strengths and limitations

Emerging parallelism models offer breakthrough scalability by fundamentally reducing inherent state and computation dependencies. They demonstrate strong compatibility with complex smart contracts. Adaptive variants of these models exhibit remarkable adaptability to diverse transaction patterns. But it requires significant ecosystem changes, including potential modifications to existing smart contract languages and execution engines. Furthermore, many emerging models remain in early stages, with limited real-world testing at scale. The complexity of these models can introduce new challenges in terms of implementation and security validation.

Emerging parallelism models represent the next frontier in blockchain scalability, offering innovative solutions to longstanding bottlenecks. Their success depends on balancing innovation with practical adoption, particularly through interoperability with existing systems and robust security validation. These models hold the promise of significantly enhancing blockchain performance, but they also require careful consideration of their impact on the broader blockchain ecosystem.

6. Intra-node parallelism solutions

In this section, we delve into specific solutions that have been proposed and implemented to address intra-node parallelism in blockchain systems. These solutions are categorized based on the parallelism models discussed earlier: deterministic, optimistic, and emerging. Each category includes a detailed examination of representative solutions, highlighting their core mechanisms, performance outcomes, use cases, and limitations. The transaction flow comparison across three models is shown in Fig. 6.

Fig. 6 extends the parallel execution framework in Fig. 2 by comparing transaction flows across the three intra-node models, highlighting model-specific differences in pre-processing and validation steps.

6.1. Deterministic parallelism solutions

Deterministic parallelism solutions prioritize upfront conflict prevention through dependency analysis, ensuring that parallel execution produces results identical to sequential processing. These solutions are widely adopted in networks where consistency and predictability are critical. Some typical deterministic parallel solutions are shown in Table 2.

Solana [18,19] enables high-throughput real-time applications via GPU-accelerated intra-node parallelism. Its Turbine Protocol fragments blocks into 64 KB chunks for parallel propagation across the P2P network. The Gulf Stream pre-executes independent transactions, and compresses state access patterns across incoming transactions, grouping those with non-overlapping sets for parallel execution on multi-core CPUs. It achieves 65k TPS by leveraging 128+ GPU cores, though performance drops to 45k TPS in high-conflict DeFi due to strict account isolation limits.

In Sui [45], the Object-Oriented State Model enforces parallelism via strict object ownership rules. "Owned objects" can only be modified by one transaction at a time, while "mutably shared objects" trigger dynamic pre-execution checks. During pre-execution, Sui's engine tracks which transactions access shared objects, ordering only conflicting ones sequentially, allowing all other transactions to run in parallel across cores.

PEEP [46] supports backward-compatible Ethereum scaling without EVM modifications. The Hybrid Static-Dynamic Dependency Graph combines two layers. Static analysis parses EVM bytecode to pre-identify potential read/write sets, while dynamic tracking records actual state accesses during a lightweight pre-execution. This hybrid approach reduces false dependencies compared to pure static analysis, unlocking more parallel batches for execution. It reduces false dependencies by 30 % vs. pure static analysis, enabling 14k TPS for ERC-20 transfers. It is critical for networks needing to retain existing contracts.

DMVCC [20] enhances parallelism in enterprise permissioned blockchains via fine-grained state control. Write versioning eliminates 99 % of write-write conflicts, delivering 20x serial execution throughput, but requires 2.3x more RAM (16GB vs. 7GB) to maintain multi-version state. It is justified for enterprise hardware budgets.

MVTO [47] (Multi-Version Transaction Ordering) enables deterministic parallelism via multi-version state tracking and conflict-serializable scheduling. Miners first identify a conflict-free execution order using database-style concurrency control, then embed transaction write sets and their serial priority into the block. Validators build version chains for state variables and route transactions to parallel threads. Non-conflicting transactions run concurrently, while conflicting ones follow the pre-defined version order to avoid state inconsistencies. This design ensures deterministic outcomes across nodes.

ParBlockchain [48] introduces OXII, which is a new paradigm for permissioned blockchains to support distributed applications that execute concurrently. It first parses transaction read/write sets to build a DAG, then partitions the DAG into independent batches via traversal. Batches with no inter-dependencies are assigned to separate threads for parallel execution. Within batches, a lightweight lock manager enforces sequential runs only for conflicting transactions. A global commit log verifies post-execution state consistency against the DAG-derived serial order, making it suitable for permissioned blockchains supporting DApps.

Yu et al. [49] use multi-thread technology to implement the proposed model where transactions are executed in parallel. A transaction splitting algorithm is proposed to resolve the synchronization problem. It splits transactions into smaller, parallelizable sub-tasks and distributes sub-tasks across threads. A dedicated synchronization module tracks

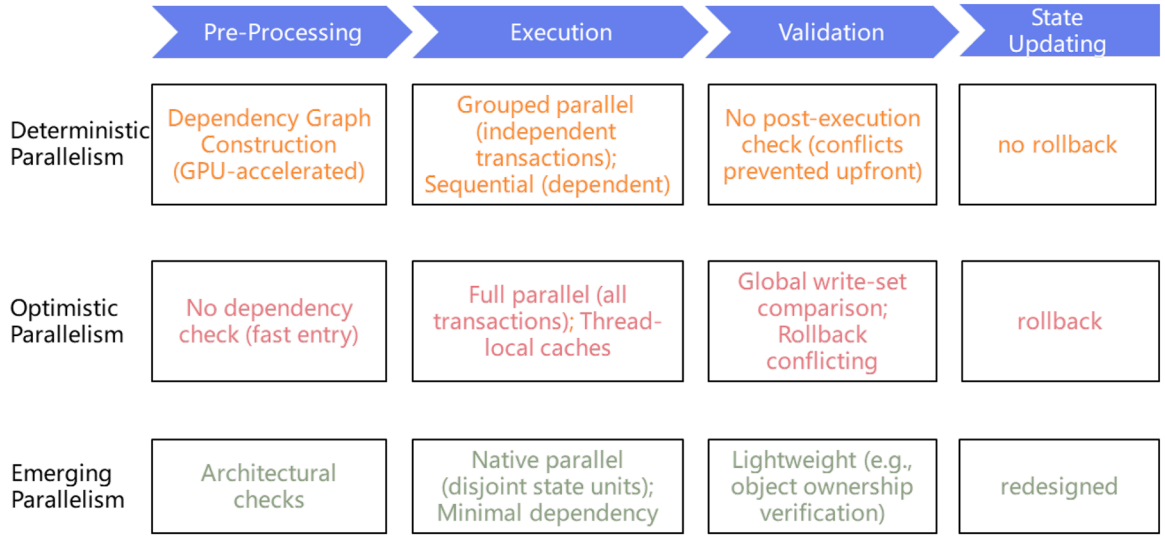

Fig. 6. Transaction flow comparison: deterministic vs. optimistic vs. emerging parallelism.

Table 2
Typical deterministic parallel solutions.

Solutions	Year	Core	Performance	Use Case	Limitations
Solana [18,19]	2017	Turbine Protocol; Gulf Stream	65k TPS	high-frequency trading, real-time apps	relies on strict account isolation; struggles with complex contracts with shared state
Sui [45]	2022	Object-Oriented State Model	297k+ TPS	NFTs, asset-centric DApps	requires object-based contract design; incompatible with traditional accounts
PEEP [46]	2021	Static+Dynamic Dependency Graph	14k+ TPS	Ethereum scaling (backward-compatible)	static analysis overestimates dependencies, limiting parallelism gains
DMVCC [20]	2023	Distributed Multi-Version CC	20x serial execution	enterprise permissioned blockchains	high memory overhead from maintaining multiple state versions
MVTO [47]	2018	Multi-Version Transaction Ordering	2.5x serial speedup(3 threads)	Ethereum 2.0 scalability	complex integration with existing EVM infrastructure
ParBlock-chain [48]	2019	OXII paradigm	–	permissioned blockchains to support DApps	requires a trusted environment due to reliance on node identities
[49]	2018	transaction splitting algorithm	remarkable development	real-time requirements in some situations	may not handle dynamically generated data access patterns well.

sub-task dependencies and uses fine-grained locks to prevent race conditions. This approach avoids monolithic transaction serialism and improves throughput for real-time-required permissioned blockchain scenarios.

6.2. Optimistic parallelism solutions

Optimistic parallelism solutions prioritize maximum parallelism by deferring conflict resolution to post-execution, making them effective in low-conflict environments. Table 3 shows some typical optimistic parallelism solutions.

Aptos (Block-STM) [16,17] achieves general-purpose high throughput for consumer DApps. Thread-local state caches enable 100k TPS in low-conflict scenarios, but rollback rates spike to 12 % in high-conflict DeFi, reducing effective throughput to 12k TPS. This aligns with its focus on low-to-medium conflict workloads. Software Transactional Memory uses thread-local state caches. Each core executes transactions against a private copy of the state, avoiding locks during execution. Post-execution, a global validation step compares write sets across all caches. If two transactions modified the same state variable, the later transaction is rolled back. Block-STM optimizes this by batching validation checks, reducing the overhead of cross-thread comparisons.

Monad [50] optimizes EVM-compatible high-performance smart contracts with minimal rollback overhead. Lightweight locks reduce validation latency by 40 % vs. Aptos, achieving 10k TPS for DeFi swaps, though adversarial high-conflict transactions still trigger 8 % rollbacks. It targets networks balancing speed and EVM compatibility. Optimistic Concurrency Control implements lightweight speculative locks. Transactions acquire temporary locks on state variables during execution, but only validate these locks at commit time. If a lock conflict is detected, the transaction with the lower priority is rolled back. It minimizes the time spent on invalid execution compared to traditional OCC.

OCCDA [51] is an optimistic concurrency control scheduler with deterministic aborts, enabling OCC scheduling in public blockchain settings. It uses partitioned counters and special commutative instructions to break up the application conflict chains in order to maximize the potential speedup. The deterministic abort mechanism uses partitioned counters to track historical conflict patterns. When scheduling new batches, OCCDA assigns transactions with high historical conflict rates to separate threads, reducing rollback rates while maintaining optimistic execution's low pre-execution latency.

OptSmart [52] optimizes optimistic execution via AU (Atomic Unit) classification and STM-based concurrency. It first splits smart contract logic into independent AUs and groups non-dependent AUs into a "concurrent bin" for parallel execution via optimistic Software Transactional Memory (STM). Dependent AUs are organized into a "block graph (BG)" and executed sequentially post-validation. A concurrent validator re-runs AUs using the same bin/BG structure to ensure determinism, achieving 5.21x–14.96x serial throughput for EVM-compatible networks.

RapidLane [53] is an extension for parallel execution engines that allows the engine to capture computations in conflicting parts of transactions and defer their execution until a later time, sometimes optimistically predicting execution results. This technique, coupled with support for a new construct for smart contract languages, allows one to turn certain sequential workloads into parallelizable ones. It identifies conflict-prone code segments during execution, defers their execution to a post-validation phase, and optimistically predicts their outcomes to keep non-conflicting code running in parallel. Post-execution, it validates predicted results against actual computations, only rolling back transactions with mismatches. This reduces wasted parallel effort on conflicting logic.

Qi et al. [54] propose a smart contract model that optimistically

Table 3
Typical optimistic parallel solutions.

Solutions	Year	Core	Performance	Use Case	Limitations
Aptos (Block-STM) [16,17]	2022	Software Transactional Memory	100k+ TPS	General-purpose DApps, DeFi	high rollback costs in high-conflict scenarios (e.g., popular DeFi contracts)
Monad [50]	2022	Optimistic Concurrency Control	10k+ TPS	High-performance smart contracts	relies on low conflict rates; adversarial transactions can trigger rollbacks
OCCDA [51]	2022	Optimistic CC with deterministic aborts	18x serial execution	mixed-workload networks	pre-analysis adds latency; prediction accuracy impacts performance
OptSmart [52]	2024	optimistic (STMs)	5.21x-14.96x serial	EVM-compatible networks	limited to known contract patterns; new contracts may trigger inefficiencies
RapidLane [53]	2024	capture computations in conflicting parts	12x	integrated into Block-STM; deployed on Aptos	limited complex apps
[54]	2023	latency-first smart contract model	greatly reduce latency	low latency	extra programming tool
PaVM [55]	2024	VM supporting inter/intra-contract parallel execution	33.4x	inter/intra-contract parallel execution	not yet in practical use

accepts committed transactions allows users to change the contract state in advance and "promise" to prove a transaction at a later time. An unproved transaction causes a revocation that rolls back the contract state. In the latency-first smart contract model, a contract has independent states managed by users. It allows users to modify contract state in advance and splits contract state into user-specific independent units, so transactions only depend on their owner's state, not unrelated commitments. This allows future transactions not to rely on unrelated transaction commitments, thereby reducing the influence of revocations.

PaVM [55] is the first smart contract virtual machine that supports both inter-contract and intra-contract parallel execution to accelerate the validation process. It consists of key instructions for precisely recording entire runtime information at the instruction level, a runtime system with a re-designed machine state and thread management to facilitate parallel execution, and a read/write-operation-based receipt generation method. It adds custom instructions to record runtime state access at the instruction level, then uses a re-designed runtime system to split contract logic into parallelizable intra-contract tasks and inter-contract calls. A read/write-based receipt generator validates post-execution operation correctness and blockchain data consistency.

6.3. Emerging parallelism solutions

Emerging parallelism solutions leverage novel state models or computational paradigms to enable parallelism, often redefining core blockchain components. Table 4 provides some emerging parallelism solutions.

Kindelia [56] is a peer-to-peer functional computer capable of hosting applications that stay up forever. It uses the HVM, a blazingly fast functional virtual machine, to run formally verified apps natively, making it as secure as mathematically possible. It replaces expensive Merkle tree insertions by reversible heaps snapshots, greatly reducing the cost of SSTORE and SLOAD, which makes real-time apps economically viable. The parallelism is inherent to its computation model: functional programs are decomposed into independent interaction net nodes, which the VM distributes across cores seamlessly. Its design prioritizes both performance and formal correctness, making it a foundational framework for parallelizable decentralized applications.

While UTXO (Unspent Transaction Output) [61] models are traditionally sequential, modern extensions enable parallelism by treating UTXOs as independent state units. Fuel [57] extends UTXO models to support smart contracts. Each UTXO in Fuel acts as a self-contained state unit, storing value, contract bytecode, and constraints. This enables transactions interacting with disjoint UTXOs to execute in parallel without dependency analysis. Its FuelVM further optimizes performance by processing independent UTXO chunks across threads and using lightweight cross-UTXO validation for interdependent transactions. A key optimization removes zero-value outputs to minimize state bloat, while its Sway language simplifies building parallelizable DApps. Fuel achieves high throughput by eliminating global state bottlenecks, outperforming traditional account-based systems in high-concurrency scenarios.

Cardano [60] leverages the Extended UTXO (eUTXO) model to enable secure, predictable parallel execution for smart contracts. Building on Bitcoin's branch-based UTXO design, Cardano achieves parallelism by splitting DApp logic across multiple UTXOs. Smart contracts manage state through sets of independent UTXOs and off-chain metadata, allowing multiple users to interact with the same contract without concurrency failures. While individual UTXO branches require sequential validation, disjoint branches execute in parallel within a block's budget. The eUTXO model also ensures execution cost predictability, avoiding the gas surprises of account-based systems. Cardano's parallelism is further enhanced by design patterns like semaphores, which coordinate access to shared resources without sacrificing performance.

Linera [58] revolutionizes scalability with its microchain architecture, partitioning the network into user- or application-specific lightweight sub-chains that run in parallel across node cores. Each microchain processes its owner's transactions sequentially, but hundreds of microchains operate concurrently, eliminating cross-user conflicts. Cross-microchain interactions are handled via asynchronous batched messaging, which avoids blocking parallel execution. Linera's scheduler dynamically allocates cores to active microchains, ensuring low latency for high-traffic applications like real-time auctions. This break from the "single global chain" paradigm enables millisecond response times, making it ideal for interactive DApps like games and social platforms.

Some frameworks use machine learning [62] to predict transaction conflict probabilities. Based on predictions, the engine allocates transactions to deterministic or optimistic execution paths. This hybrid approach balances latency and rollback overhead, adapting to varying network conditions. Sei V2 [59] advances parallel execution with its Twin-Turbo architecture, combining adaptive parallelism and optimized consensus to deliver high throughput and low latency. Its core innovation is dynamic switching between two execution engines: an Optimistic Engine for low-conflict transactions that skips pre-analysis and uses thread-local caches, and a Deterministic Engine for high-conflict workloads that builds incremental dependency graphs. This adaptability is paired with Twin-Turbo consensus, which uses smart block propagation (validating transaction hashes from memory pools) and optimistic block processing (executing before final consensus) to cut latency. Sei V2 adds EVM compatibility via pointer contracts and SeiDB to reduce state bloat, positioning it as a high-speed L1 for digital asset exchanges and parallelized DApps.

7. Discussion

The discussion section aims to provide a comprehensive analysis of the applicability, trade-offs, and holistic execution optimization strategies related to intra-node parallelism in blockchain systems. This section synthesizes the findings from the previous sections and offers practical insights for selecting and implementing parallel execution solutions.

Table 4
Typical emerging parallel solutions.

Solutions	Year	Core	Performance	Use Case	Limitations
Kindelia [56]	2022	high-order VM	1-second block time	real-time applications	limited real-world deployment and ecosystem maturity
Fuel [57]	2020	UTXO with parallelism	–	scalable DeFi, privacy-focused apps	complex cross-UTXO transactions; limited tooling compared to account models
Linera [58]	2023	microchain parallelism	–	high-scale, user-centric apps	unproven in large decentralized networks
Sei V2 [59]	2023	Twin-Turbo parallelism	400 ms time to finality	DeFi, order-book exchanges	complex mode-switching logic; potential latency in high-conflict transitions
Cardano [60]	2017	extended UTXO model	–	supporting multi-assets and smart contracts	complex eUTXO model

7.1. Applicability analysis

The effectiveness of a parallel execution solution depends heavily on the transaction characteristics and network requirements of its target scenario. There are some key factors guiding selection.

1. Conflict Density. High-conflict environments benefit from deterministic models. These models prevent rollbacks by pre-identifying dependencies, avoiding wasted resources on invalid parallel execution. In contrast, low-conflict scenarios thrive with optimistic models, where minimal conflicts make rollback overhead negligible.
2. Contract Complexity. Complex smart contracts with dynamic state access challenge static dependency analysis in deterministic models. Here, optimistic or emerging models offer flexibility, as they avoid overestimating dependencies. For simple, standardized contracts, such as ERC-20 transfers, deterministic solutions suffice and reduce complexity.
3. Ecosystem Compatibility. Networks requiring backward compatibility with existing infrastructure, like Ethereum, prioritize solutions like PEEP or OCCDA, which integrate with the EVM without overhauling contract languages. New blockchains can adopt emerging models to prioritize long-term scalability over compatibility.
4. Latency Requirements. Latency-sensitive applications, such as real-time supply chain tracking, favor optimistic models, which avoid pre-execution dependency checks. Applications prioritizing predictability, like financial settlements, prefer deterministic models, where execution times are more consistent.
5. Backward Compatibility with EVM Ecosystems. (1) Deterministic solutions: Work with most of existing ERC-20/ERC-721 contracts by parsing EVM bytecode for dependency graphs, and no contract rewrites needed. (2) Optimistic solutions: Support unmodified DeFi contracts via EVM-compatible caches, but complex contracts have higher rollback rates. (3) Emerging solutions: Require contract refactoring, only a small number of existing EVM DApps are compatible.

7.2. Overhead trade-offs

While parallel execution enhances throughput, it introduces unavoidable overhead that must be managed to avoid negating performance gains.

1. Computational Overhead. Deterministic models incur costs from dependency graph construction and thread synchronization. For large blocks, graph construction can become a bottleneck, especially with complex contracts. Optimistic models face rollback and validation costs. In high-conflict scenarios, rollbacks can reduce throughput below serial levels.
2. Resource Overhead. Parallel execution increases memory usage, as solutions like DMVCC (multi-version state) or Block-STM (thread-local caches) maintain temporary state copies. This can strain node hardware, raising barriers to entry for decentralized participation. Networked nodes must synchronize parallel execution results, introducing communication overhead in cross-node validation.

Theoretical maximum throughput (TPS) for intra-node parallelism depends on two core variables: number of CPU cores (n) and transaction conflict rate (c%). Below are simplified bounds derived from queueing theory and parallel computing models, validated against empirical data from solutions like Solana (deterministic) and Aptos (optimistic).

1. Deterministic Parallelism Bounds. Deterministic models split transactions into independent batches (constrained by c%) and distribute batches across n cores. Pre-execution dependency analysis ($T_{analysis}$, time per transaction) adds fixed overhead. For N_{trans} transactions, maximum parallel batches per block is $\min(n,\ N_{trans} \times (100 - c)\%)$, capped by core count or independent transactions. Total time per block $Time_{total}$ is:

$$Time_{total} = \frac{N_{trans} \times (100 - c)\%}{\mathrm{n}} \times T_{exec} + N_{trans} \times T_{analysis} \tag{1}$$

Where T_{exec} is average transaction serial execution time. The theoretical TPS upper bound is:

$$TPS_{upper} = \frac{N_{trans}}{Time_{total}} \tag{2}$$

This aligns with Solana's real-world performance: with $n = 128$ (GPU-accelerated "virtual cores"), $c\% = 5\%$ (low-conflict), $T_{exec} = 1ms$, $T_{analysis} = 0.01ms$, TPS_{upper} jumps to ~57,471 TPS, matching its 65k TPS claim. The hard limit: As n exceeds the number of independent batches $N_{trans} \times (100 - c)\%$, adding cores no longer boosts TPS (diminishing returns). When all transactions conflict (c approaches 100), TPS_{upper} approaches serial TPS, and is $N_{trans}/(T_{exec} + T_{analysis})$.

2. Optimistic Parallelism Bounds. Optimistic models prioritize execution, so their upper bound depends on rollback cost and validation overhead. Rollback efficiency is r. Time for initial parallel execution is $T_{init} = T_{exec} \times Max(1,\ N_{trans}/n)$, capped by single-core time if $N_{trans} < n$. Time for rollback and re-execution is $T_{back} = N_{trans} \times c\% \times T_{exec} \times (1 - r)$.

Total block time $Time_{total}$ is:

$$Time_{total} = T_{init} + T_{back} \tag{3}$$

When $n = 128$, $c\% = 5\%$, $r = 0.8$, $T_{exec} = 1ms$, TPS_{upper} is 56,180. It is lower than Aptos' 100k TPS. As c approaches 0, T_{back} is 0, and TPS_{upper} is 128k, which is matches Aptos' 100k TPS.

7.3. Holistic execution optimization

Parallel transaction execution is one lever for improving execution layer efficiency. There are other strategies to maximize the performance of blockchain systems.

1. Reducing State I/O Overhead. Transaction throughput is often bottlenecked by database read/write latency, not just CPU limits. Optimizations like state compression, caching hot state, and batch processing directly boost efficiency. For example, Fuel's UTXO+ model reduces I/O by isolating state in independent UTXOs, limiting disk operations per transaction.
2. Execution Engine Optimization. Enhancing the underlying virtual machine can reduce per-transaction computation time. Examples include just-in-time (JIT) compilation, gas-efficient instruction sets, and precompiled contracts.
3. Transaction Batching. Grouping logically related transactions [63] allows shared state access checks, reducing redundant dependency analysis. Hyperledger Fabric's batch processing and Sei's order-book batching leverage this to reduce per-transaction overhead.

7.4. Chain-level scalability

While execution layer parallelism is critical, end-to-end blockchain scalability depends on technical synergies between intra-node parallelism and other layers.

1. Consensus Layer Optimization. Slow consensus limits how many transactions can be processed, regardless of execution parallelism.

Innovations like PoS [64], sharded consensus, and DAG (Directed Acyclic Graph)-based consensus [65] increase throughput by accelerating block finality and reducing bottlenecks in transaction ordering.

Intra-node parallelism enhances PoS in three technical ways. (1) Pre-Execution During Block Proposal. Instead of waiting for full block ordering, validators use intra-node optimistic parallelism to pre-execute transactions in parallel while the consensus layer finalizes the block order. This "overlapping" of pre-execution and consensus reduces end-to-end latency. (2) Dependency Graph Alignment with Validator Sharding. Transactions in a shard are pre-analyzed to identify independent groups, which are executed in parallel across the shard's validators. This reduces cross-shard communication overhead. (3) Rollback Mitigation via Stake Incentives. For optimistic intra-node models, this incentive aligns with parallel execution. Validators use historical conflict data to avoid including high-conflict transaction groups in their proposals.

Intra-node parallelism complements DAG through (1) Intra-Node Parallel Validation of DAG Tips. DAG networks process "tips" (unvalidated transaction nodes) in parallel. Intra-node dynamic pre-execution is used to validate multiple tips simultaneously. Each tip's read/write set is tracked in real time, and non-conflicting tips are executed in parallel across CPU cores. (2) Conflict Resolution Alignment with DAG Topology. DAGs resolve conflicts via transaction weight. Intra-node parallelism enhances this by tagging conflicting transactions with their rollback cost metrics during validation. The DAG consensus layer prioritizes transactions with lower rollback costs, reducing the likelihood of invalidating parallel-executed transactions.

2. Network Layer Efficiency. Transaction propagation delays can starve execution engines of input, even with parallel capabilities. Protocols like Solana's Turbine (block fragment propagation) and Linera's microchain gossiping optimize P2P communication, ensuring transactions reach execution engines faster.

Solana's Turbine protocol splits blocks into 64 KB fragments, which are propagated in parallel across the P2P network. Intra-node parallelism integrates with Turbine via: (1) Fragment-Level Parallel Execution. As fragments arrive at a node, the intra-node deterministic engine immediately parses each fragment to extract transaction read/write sets. Independent transactions across fragments are grouped for parallel execution. (2) Dependency Resolution Across Fragments. Turbine fragments may contain dependent transactions. The intra-node dynamic analysis layer tracks cross-fragment dependencies in real time, pausing execution of dependent transactions until all required fragments are received.

Linera splits the network into microchains and uses gossiping to propagate microchain transactions. (1) Microchain-Specific Parallel Engines. Each microchain in Linera has a dedicated intra-node execution thread. (2) Asynchronous State Sync with Parallel Execution. Linera's network layer syncs microchain states asynchronously. Intra-node emerging parallelism executes transactions in a microchain while its state is syncing, using versioned state snapshots to avoid conflicts.

3. Practical Implications of Layer Synergy. The technical mappings above highlight a critical insight: intra-node parallelism is not a standalone optimization but a layer-integrated enabler of scalability. PoS networks gain the most from optimistic intra-node models (pre-execution during proposal), while DAGs benefit more from dynamic parallel validation (tip processing). Block fragmentation protocols (Turbine) require deterministic intra-node models to handle cross-fragment dependencies, while microchain gossiping (Linera) works best with emerging parallelism (per-microchain engines). Without this synergy, even advanced intra-node models underperform.

7.5. Balancing scalability, security, and decentralization

Scalability gains via parallel execution must not come at the expense of blockchain's core properties. Some parallel models face the challenge of decentralization. Parallel execution often demands more powerful hardware to handle overhead, raising barriers to entry for small-scale nodes. This risks centralization, as only well-resourced entities can participate. Solutions like lightweight parallel engines (e.g., Canto's optimized EVM [66]) or hardware-agnostic scheduling (adapting parallelism to node capabilities) help mitigate this. Blockchain systems must prioritize based on use case. Permissioned blockchains may accept slightly reduced decentralization for higher throughput, while permissionless networks must strictly balance security and scalability to maintain trust. Intra-node parallelism boosts throughput but introduces unique security risks, tied to how each model handles conflicts, state, or execution.

Deterministic models rely on pre-execution dependency checks, so attacks target these checks to break parallelism or cause inconsistencies. (1) False Dependency Injection. Attackers lie about which state variables a transaction uses. This tricks the dependency graph into grouping independent transactions sequentially, cutting throughput. Mitigation: Cross-check declared state access with real-time usage during pre-execution. In permissioned networks, penalize users who submit false data. (2) Dependency Graph Poisoning. Attackers use hidden state accesses. This makes the graph miss conflicts, leading to inconsistent states. Mitigation: Use symbolic execution to uncover all possible state accesses. Add quick post-execution checks: compare parallel results to a small serial snapshot for high-risk transactions.

Optimistic models execute first, then validate, so attacks exploit rollbacks or validation delays. (1) Rollback Amplification. Attackers send batches of conflicting transactions. This triggers mass rollbacks, wasting resources and crashing throughput. Mitigation: Switch to deterministic mode if rollbacks exceed a threshold. Group transactions by conflict risk to isolate high-conflict batches. (2) Validation Race Conditions. Attackers exploit delays between execution and validation to overwrite state. Mitigation: Lock state variables during validation. Add timestamps to state versions, and reject transactions if a newer version exists during validation.

Emerging models redefine state, creating new vulnerabilities in untested designs. (1) Object Ownership Bypass. Attackers forge proof of owning an object to modify others' assets. Mitigation: Use formal math proofs to verify ownership rules. Add object-level permission checks that ignore tampered metadata. (2) UTXO Dependency Confusion. Attackers hide shared dependencies in UTXOs. The engine treats them as independent, causing inconsistencies. Fuel testnets had balance errors from this. Mitigation: Analyze UTXO scripts for hidden dependencies. Check for overlapping script references across parallel UTXOs.

7.6. Deployment for intra-node parallelism

(1) Backward Compatibility. EVM-compatible networks: Deterministic (MVTO) and optimistic (Aptos Block-STM) models integrate with most of existing contracts, and only minor gas adjustments may be needed. Emerging models: Kindelia (functional programming) and Cardano (eUTXO) require full contract rewrites.
(2) Node Hardware Specifications. CPU: 8+ cores for deterministic (Solana) or 16+ cores for optimistic (Aptos) models, and 4-core CPUs cause throughput loss. RAM: 16GB+ for multi-version state (DMVCC) or 32GB+ for emerging models (Sui), and insufficient RAM leads to execution failures. Storage: SSDs required for deterministic models to avoid I/O bottlenecks, and HDDs cut throughput.
(3) Validator Participation. Barriers to entry: Serial nodes run on consumer hardware, but parallel nodes need larger RAM, reducing small-scale participation. Cost and incentives: Parallel

nodes have higher electricity use. Without reward boosts, small validators may plan to exit.

8. Comparison, benchmarking, limitations, and trend

This section systematically compares the three core intra-node parallelism models—deterministic, optimistic, and emerging—across key dimensions, including design philosophy, conflict handling, performance, and practical trade-offs. Additionally, it outlines future trends in intra-node parallelism, highlighting potential advancements and areas of research.

8.1. Comparison

To provide a clear overview of intra-node parallel execution models and their practical implications, we present a detailed comparison in Table 5. This comparison highlights the fundamental differences between the three parallelism models.

Deterministic Parallelism: Best suited for environments where transaction dependencies can be reliably identified upfront. This model ensures consistency and predictability, making it ideal for enterprise solutions and high-conflict scenarios. However, it may introduce latency due to pre-execution analysis and may not fully leverage parallel hardware in complex scenarios.

Optimistic Parallelism: Maximizes throughput by allowing transactions to execute concurrently and resolving conflicts post-execution. This model is highly effective in low-conflict environments and latency-sensitive applications. However, it can suffer from significant rollback overhead in high-conflict scenarios, potentially negating performance gains.

Emerging Parallelism: Represents the cutting edge of intra-node parallelism, offering innovative solutions that fundamentally redesign state models and computation paradigms. These models can achieve significant performance gains but often require substantial changes to existing systems and may introduce new complexities and security considerations.

Table 5
Comparison of differences between the three parallelism models.

Dimension	Deterministic Parallelism	Optimistic Parallelism	Emerging Parallelism
Core Idea	prevent conflicts upfront via dependency analysis	maximize parallelism by executing first, then resolving conflicts	redesign state/computation paradigms to enable native parallelism.
Conflict Handling	static/dynamic dependency detection; transactions are grouped	without prior checks; roll back and re-execute serially	conflicts minimized via architectural design or fine-grained state partitioning
Throughput	2–3x serial execution	5–10x serial execution	varies widely
Rollback Rate	<1 % (negligible)	5–20 %	<5 %
Latency	100–300 ms per block	50–150 ms per block	40–200 ms per block
Scenarios	high-conflict environments; enterprise solutions	low-conflict; latency-sensitive apps	complex DApps; new blockchains with significant architectural changes
Complexity	moderate: relies on dependency graph construction and analysis tools	high: requires robust rollback mechanisms, temporary state management, validation coordination	very high: demands redesign of state models, execution engines, etc.
Solutions	Solana, Sui, PEEP, DMVCC, MVTO, ParBlockchain	Aptos, Monad, OCCDA, OptSmart, RapidLane, PaVM	Kindelia, Fuel, Linera, Sei V2, Cardano

No single model dominates across all scenarios, and the optimal choice depends on a network's specific requirements. Choose deterministic parallelism for high-conflict, predictable workloads or enterprise networks prioritizing compatibility and low rollback risk. Choose optimistic parallelism for low-conflict, latency-sensitive applications where maximum throughput in ideal conditions outweighs occasional rollback costs. Choose emerging parallelism for new blockchains or complex DApps willing to adopt new paradigms to achieve long-term scalability, accepting short-term adoption challenges. This diversity underscores the importance of hybrid approaches (discussed in Section 8.2) that combine strengths across models to address the limitations of single-model solutions.

8.2. Standardized benchmarking

Inconsistent evaluation metrics and testbeds have long hindered reliable comparisons of intra-node parallelism solutions. This section clarifies essential benchmarking metrics to guide fair assessment and outlines key open challenges in establishing standardized evaluation practices for intra-node parallelism.

8.2.1. Essential benchmarking metrics

The core metrics capture the performance, efficiency, and practical viability of intra-node parallelism solutions, addressing gaps in disjointed evaluation across prior studies.

1. Throughput
   The number of valid transactions executed and committed per second, measured under controlled conflict rates. TPS alone is incomplete, so it must be tied to workload characteristics (transaction complexity, conflict density) for meaningful comparison.
2. Rollback Rate
   The percentage of parallel-executed transactions aborted and re-executed due to unresolved conflicts. Critical for optimistic and emerging models, yet this metric is often omitted in existing TPS-focused analyses.
3. Parallel Efficiency
   The ratio of actual parallel throughput to the theoretical maximum, quantifying multi-core hardware utilization. This metric exposes how well solutions leverage available hardware.
4. Resource Overhead
   Additional CPU, memory, or I/O consumed by parallelism mechanisms vs. serial execution. Key sub-metrics: CPU load increase, memory overhead, and I/O latency. High overhead risks centralization by pricing out small-scale nodes.

8.2.2. Challenges in standardized benchmarking

The unresolved challenges prevent consistent evaluation of intra-node parallelism, representing priority areas for future work.

1. Workload Standardization: Existing studies use arbitrary transaction mixes, making cross-solution comparisons invalid. A standardized suite should include low/medium/high-conflict scenarios to reflect real-world use cases.
2. Controlled Conflict Generation: No agreed method exists to simulate natural conflicts. Current approaches are unrealistic, and future tools need to mimic real-world access patterns.
3. Decentralization-Aware Metrics: Benchmarks focus on performance but ignore accessibility. A new metric is needed to balance scalability and inclusivity.
4. Dynamic Workload Support: Real blockchains face sudden workload shifts, but benchmarks use static conflict rates. Tools must simulate dynamic changes to test how solutions adapt.

8.3. Limitations

While this survey provides a structured analysis of intra-node parallelism, it has three key limitations that guide future research directions.

(1) Empirical Depth Constraints: The analysis synthesizes performance data from published studies and technical documentation, rather than conducting original experimental validation. This reliance on secondary data introduces variability in results and limits insights into edge-case behavior.
(2) Emerging Model Coverage Gaps: While this survey includes representative emerging solutions, it excludes nascent paradigms (e.g., AI-driven adaptive parallelism, quantum-resistant parallel execution) due to their limited published data. These emerging approaches may redefine intra-node parallelism but were deemed too preliminary for rigorous analysis, leaving a gap in forward-looking coverage.

8.4. Future trend

As intra-node parallel execution research matures, future advancements will focus on addressing remaining trade-offs and leveraging emerging technologies to create more robust, adaptive, and holistic solutions.

1. Hybrid Parallelism

Future solutions will dynamically combine deterministic, optimistic, and emerging strategies based on real-time transaction characteristics. Machine learning will be integrated to predict conflict probabilities, automatically routing transactions to the optimal execution path. Hybrid models will offer the best of all worlds, balancing the strengths of each approach to achieve higher performance and efficiency. For example, a lightweight LSTM (Long Short-Term Memory) model could analyze past 24-hour transaction patterns to route transactions to the optimal engine: deterministic for predicted high-conflict batches and optimistic for low-conflict ones. A prototype of this approach, tested on Aptos testnets [46], reduced rollback rates compared to static mode switching.

2. Hardware-Aware Parallel Execution

Current solutions underutilize modern hardware architectures. Future work will focus on hardware-software co-design to maximize parallel efficiency, including multi-core optimization, specialized accelerators, lightweight state storage [67]. By leveraging the full potential of modern hardware, these solutions will significantly enhance transaction throughput and reduce latency.

Future progress will hinge on deeper hardware-software co-design. (1) GPU-Accelerated Execution. GPUs excel at parallel, data-intensive tasks, making them ideal for dependency graph construction and batch transaction execution, yet only a few solutions like Solana leverage them today. However, challenges persist. GPUs struggle with irregular workloads, leading to underutilization. Future work will focus on "hybrid GPU-CPU pipelines". offloading repetitive tasks to GPUs, while reserving CPUs for dynamic, logic-heavy execution. (2) FPGA/ASIC-Based Optimization. FPGAs (field-programmable gate arrays) and ASICs (application-specific integrated circuits) offer faster, more energy-efficient execution than general-purpose CPUs, but their adoption in intra-node parallelism is limited. FPGAs can be customized for parallelism tasks like conflict detection. ASICs, while faster, lack flexibility, they work only for fixed workloads. The key challenge is balancing customization (for speed) and adaptability (for diverse smart contracts). Future designs may use "partially reconfigurable FPGAs" that adjust to workload changes.

Even with specialized hardware, poor scheduling negates gains. Critical challenges emerge: (1) Workload Partitioning Ensuring tasks match hardware capabilities. (2) Memory Latency: GPUs/FPGAs often face delays accessing shared state.

3. Automated Dependency Management

Manual dependency analysis via static/dynamic tools is error-prone and resource-intensive. Future solutions will adopt self-optimizing dependency management through AI-driven dependency prediction and smart contract annotation. Automated dependency management will reduce the overhead associated with conflict detection and improve the scalability of intra-node parallelism. For example, a Solidity plugin could scan Uniswap V4 code, automatically tag independent functions with //@parallel annotations, and flag shared state variables for deterministic handling.

4. Holistic Execution Layer Reinvention

Beyond incremental optimizations, future trends will revisit core execution layer assumptions to enable native parallelism. This includes event-driven state updates, immutable intermediate states [68], domain-specific virtual machines [69], serverless-like execution. These innovations will fundamentally transform the execution layer, unlocking new levels of performance and efficiency. For instance, a supply chain blockchain could model state as a stream of events. Each event is processed in parallel if it targets disjoint assets. Hyperledger Fabric's upcoming Event-STM module implements this.

5. Security and Ecosystem

As parallelism grows more complex, ensuring security and decentralization will be critical. Standardization efforts will focus on creating common frameworks and protocols to lower integration barriers and promote interoperability. Standardization will facilitate the widespread adoption of intra-node parallelism solutions, ensuring they are robust, secure, and compatible with existing blockchain ecosystems.

The future of intra-node parallel execution lies in holistic, adaptive, and resilient designs that balance performance with blockchain's core values. Hybrid models, hardware co-design, and automated dependency management will drive practical scalability gains, while state model reinvention and formal verification ensure these advancements are robust and sustainable. As the ecosystem matures, standardization and interoperability will be key to unlocking the full potential of parallelism across diverse blockchain applications.

9. Conclusion

This survey provides a comprehensive, focused analysis of intra-node transaction parallelism. It is a critical yet understudied dimension of blockchain scalability, and addresses gaps in existing literature that prioritize broad parallelism over lightweight, node-level optimizations. This work advances blockchain scalability research: (1) It establishes a systematic taxonomy of intra-node parallelism, categorizing solutions into three models (deterministic, optimistic, emerging) based on conflict-handling mechanisms and architectural intent—resolving the lack of standardized classification in prior surveys. (2) It offers an in-depth analysis of ~20 representative solutions, linking each to its underlying model and quantifying trade-offs. (3) It explores practical deployment considerations (backward compatibility, hardware requirements, validator participation) that were overlooked in prior "lightweight" optimization framing, grounding the analysis in real-world feasibility. (4) It introduces a framework for standardized benchmarking and identifies open challenges to guide consistent evaluation of intra-node solutions.

Intra-node parallelism delivers tangible benefits for both blockchain

practitioners and ecosystems, distinguishing it from resource-intensive broad parallelism. (1) For mature networks like Ethereum, deterministic and optimistic solutions enable backward-compatible scalability, boosting throughput by 2–10x without overhauling core protocols or forcing DApp rewrites. This preserves existing user bases and developer ecosystems, a critical advantage over sharding or cross-chain solutions that require architectural overhauls. (2) For new blockchains like Fuel, emerging models unlock long-term scalability, supporting complex DApps with 10–100x serial throughput while maintaining low rollback rates (<5 %). (3) Intra-node parallelism complements broad parallelism. Sharded networks can further enhance per-shard performance via node-level parallelism, creating end-to-end scalability that balances efficiency and decentralization.

To advance intra-node parallelism, three insightful directions warrant exploration. (1) AI-driven adaptive parallelism, developing machine learning models that predict transaction conflict rates in real time and dynamically switch execution modes. This would address the static nature of current models, which struggle to adapt to variable workloads. (2) Lightweight hardware-software co-design for decentralization, designing low-overhead accelerators that enable small-scale validators to run parallel execution engines, mitigating the centralization risk of current GPU/ASIC-dependent solutions. (3) Cross-layer integration frameworks, creating standardized protocols to synchronize intra-node parallelism with consensus and network layers. This would resolve ad-hoc integration and enable plug-and-play parallelism engines for diverse blockchain architectures.

By addressing these gaps and advancing these directions, intra-node parallelism can become a foundational component of high-performance, decentralized blockchain systems, meeting the scalability demands of real-world applications while preserving blockchain's core values of consistency and inclusivity.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

Supported by the Open Project Program of Anhui Engineering Research Center for Agricultural Product Quality Safety Digital Intelligence (No. FYKFKT24084 and No. FYKFKT24079), and Anhui Province Science and Technology Innovation Project (Grant No. 202423k09020016).

Data availability

No data was used for the research described in the article.

References

[1] S. Nakamoto, Bitcoin: a peer-to-peer electronic cash system, (2008). https://bitcoin.org/en/bitcoin-paper (accessed July 22, 2016).
[2] C. Piao, Y. Hao, J. Yan, X. Jiang, Privacy preserving in blockchain-based government data sharing: a service-on-chain (SOC) approach, Inf. Process. Manag. 58 (2021) 102651, https://doi.org/10.1016/j.ipm.2021.102651.
[3] A. Haleem, M. Javaid, R.P. Singh, R. Suman, S. Rab, Blockchain technology applications in healthcare: an overview, Int. J. Intell. Netw. 2 (2021) 130–139, https://doi.org/10.1016/j.ijin.2021.09.005.
[4] J. Moosavi, L.M. Naeni, A.M. Fathollahi-Fard, U. Fiore, Blockchain in supply chain management: a review, bibliometric, and network analysis, Environ. Sci. Pollut. Res. (2021), https://doi.org/10.1007/s11356-021-13094-3.
[5] C. Pan, Z. Liu, Y. Long, Research on scalability of blockchain technology: problems and methods, Comput. Res. Dev. 55 (2018) 2099–2110, https://doi.org/10.7544/issn1000-1239.2018.20180440.
[6] M.H. Nasir, J. Arshad, M.M. Khan, M. Fatima, K. Salah, R. Jayaraman, Scalable blockchains — a systematic review, Future Gener. Comput. Syst. 126 (2022) 136–162, https://doi.org/10.1016/j.future.2021.07.035.
[7] A. Hafid, A.S. Hafid, M. Samih, Scaling blockchains: a comprehensive survey, IEEE Access 8 (2020) 125244–125262, https://doi.org/10.1109/ACCESS.2020.3007251.
[8] Q. Zhou, H. Huang, Z. Zheng, J. Bian, Solutions to scalability of blockchain: a survey, IEEE Access 8 (2020) 16440–16455, https://doi.org/10.1109/ACCESS.2020.2967218.
[9] P. Barker, Parallel execution on blockchains - what you need to know, (2024). https://cryptorated.com/understanding-blockchain-parallel-execution/(accessed August 5, 2025).
[10] T. Jiang, H. Luo, K. Yang, G. Sun, H. Yu, Q. Huang, A.V. Vasilakos, Blockchain for energy market: a comprehensive survey, Sustain. Energy Grids Netw. 41 (2025) 101614, https://doi.org/10.1016/j.segan.2024.101614.
[11] H. Luo, G. Sun, J. Wang, H. Yu, D. Niyato, S. Dustdar, Z. Han, Wireless blockchain meets 6G: the future trustworthy and ubiquitous connectivity, IEEE Commun. Surv. Tutor. (2025), https://doi.org/10.1109/COMST.2025.3593918, 1–1.
[12] Introduction to smart contracts, Ethereum.Org (2025). https://ethereum.org/en/developers/docs/smart-contracts/(accessed July 22, 2025).
[13] N. Álvarez-Díaz, J. Herrera-Joancomartí, P. Caballero-Gil, Smart contracts based on blockchain for logistics management, in: Proceedings of the 1st International Conference on Internet of Things and Machine Learning, Association for Computing Machinery, New York, NY, USA, 2017, pp. 1–8, https://doi.org/10.1145/3109761.3158384.
[14] Transactions. https://ethereum.org/en/developers/docs/transactions/(accessed August 13, 2025).
[15] G. Wood, Ethereum: a secure decentralised generalised transaction ledger. (2014).
[16] Aptos: The World's Most Production-Ready Blockchain, Aptos (2025). https://aptosfoundation.org (accessed August 13, 2025).
[17] Aptos ecosystem leading the way with parallelism | Aptos. https://aptosfoundation.org/currents/aptos-ecosystem-leading-the-way-with-parallelism (accessed August 5, 2025).
[18] Web3 Infrastructure for everyone | Solana. https://solana.com (accessed August 13, 2025).
[19] A. Yakovenko, Solana: a new architecture for a high performance blockchain v0.8.13, (2022). https://solana.com/solana-whitepaper.pdf (accessed August 13, 2025).
[20] X. Qi, J. Jiao, Y. Li, Smart contract parallel execution with fine-grained State accesses, in: 2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS), 2023, pp. 841–852, https://doi.org/10.1109/ICDCS57875.2023.00068.
[21] A. Meneghetti, T. Parise, M. Sala, D. Taufer, A survey on efficient parallelization of blockchain-based smart contracts, (2019). https://doi.org/10.33166/AETiC.2019.05.002.
[22] J. Shi, C. Li, H. Wu, H. Gao, S. Jin, T. Huang, W. Zhang, Evaluating the Parallel Execution Schemes of Smart Contract Transactions in Different Blockchains: An Empirical Study, Algorithms and Architectures for Parallel Processing, Springer International Publishing, Cham, 2022, pp. 35–51, https://doi.org/10.1007/978-3-030-95391-1_3.
[23] Z. Zhang, P. Zhu, Exploring parallel blockchains: from related concepts to application scenarios, in: 2024 IEEE 4th International Conference on Digital Twins and Parallel Intelligence (DTPI), 2024, pp. 227–231, https://doi.org/10.1109/DTPI61353.2024.10778750.
[24] J. Shi, H. Wu, H. Gao, W. Zhang, Overview on parallel execution models of smart contract transactions in blockchains, J. Softw. 33 (2021) 4084–4106, https://doi.org/10.13328/j.cnki.jos.006528.
[25] FISCO BCOS. https://www.fisco.com.cn/fisco_20.html (accessed August 1, 2025).
[26] AntChainOpenLabs, GitHub. https://github.com/AntChainOpenLabs (accessed August 13, 2025).
[27] A.Z. Chahoki, M. Herlihy, M. Roveri, SoK: concurrency in blockchain – a systematic literature review and the unveiling of a misconception, (2025). https://doi.org/10.48550/arXiv.2506.01885.
[28] A. Lakhan, Z.A.A. Alyasseri, M.A. Mohammed, B. Al-Attar, J. Nedoma, R. Alubady, S. Memon, R. Martinek, Sustainable secure blockchain assisted AIoT and green multiconstraints supply chain system, IEEE Internet Things J. 12 (2025) 39392–39406, https://doi.org/10.1109/JIOT.2025.3548037.
[29] M.A. Mohammed, A. Lakhan, K.H. Abdulkareem, M.K. Abd Ghani, H.A. Marhoon, S. Kadry, J. Nedoma, R. Martinek, B.G. Zapirain, Industrial Internet of Water Things architecture for data standarization based on blockchain and digital twin technology, J. Adv. Res. 66 (2024) 1–14, https://doi.org/10.1016/j.jare.2023.10.005.
[30] G. Wood, An introduction to Polkadot, Polkadot-Light. (2020). https://assets.polkadot.network/Polkadot-lightpaper.pdf.
[31] Chainlink - the standard for onchain finance. https://www.v3-chainlink.org/.
[32] Axelar | the gateway to onchain finance. https://www.axelar.network/(accessed August 13, 2025).
[33] The liquid network. https://liquid.net/.
[34] Arbitrum - the ultimate ethereum scaling solution. https://arbtirum.cc/.
[35] Cosmos: the internet of blockchains. https://cosmos.network/whitepaper/(accessed August 13, 2025).
[36] Ethereum.Org: the complete guide to Ethereum, Ethereum.Org. https://ethereum.org/en/(accessed August 13, 2025).
[37] Zilliqa | scalable, secure & EVM-compatible blockchain for institutions. https://zilliqa.com/(accessed July 22, 2025).
[38] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, S.W. Cocco, J. Yellick, Hyperledger fabric: a distributed operating

system for permissioned blockchains, in: Proceedings of the Thirteenth EuroSys Conference, Association for Computing Machinery, New York, NY, USA, 2018, pp. 1–15, https://doi.org/10.1145/3190508.3190538.
[39] A. Hentschel, D. Shirley, L. Lafrance, Flow: separating consensus and compute, (2019). https://doi.org/10.48550/arXiv.1909.05821.
[40] A. Hentschel, Y. Hassanzadeh-Nazarabadi, R. Seraj, D. Shirley, L. Lafrance, Flow: separating consensus and compute – block formation and execution, (2020). https://doi.org/10.48550/arXiv.2002.07403.
[41] A. Hentschel, D. Shirley, L. Lafrance, M. Zamski, Flow: separating consensus and compute – execution verification, (2019). https://doi.org/10.48550/arXiv.1909.05832.
[42] Y. Lai, Y. Liu, H. Luo, G. Sun, C. Chi, H. Yu, Accelerating block and transaction propagation: a survey on broadcast protocols in blockchain networks, IEEE Trans. Netw. Sci. Eng. (2025) 1–24, https://doi.org/10.1109/TNSE.2025.3570870.
[43] Home | Solidity Programming Language. https://soliditylang.org/(accessed August 18, 2025).
[44] What is DeFi? | benefits and use of decentralised finance. https://ethereum.org/en/defi/(accessed August 14, 2025).
[45] The MystenLabs Team, The Sui Smart Contracts Platform.
[46] Z. Chen, X. Qi, X. Du, Z. Zhang, C. Jin, PEEP: a parallel execution engine for permissioned blockchain systems, in: Database Systems for Advanced Applications: 26th International Conference, 2021, pp. 341–357, https://doi.org/10.1007/978-3-030-73200-4_24.
[47] A. Zhang, K. Zhang, Enabling Concurrency on Smart Contracts Using Multiversion Ordering, Web and Big Data, Springer International Publishing, Cham, 2018, pp. 425–439, https://doi.org/10.1007/978-3-319-96893-3_32.
[48] M.J. Amiri, D. Agrawal, A. El Abbadi, ParBlockchain: leveraging transaction parallelism in permissioned blockchain systems, in: 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), 2019, pp. 1337–1347, https://doi.org/10.1109/ICDCS.2019.00134.
[49] W. Yu, K. Luo, Y. Ding, G. You, K. Hu, in: A Parallel Smart Contract Model, Proceedings of the 2018 International Conference on Machine Learning and Machine Intelligence, New York, NY, USA, 2018, pp. 72–77, https://doi.org/10.1145/3278312.3278321.
[50] Monad | the most performant EVM-compatible layer 1 blockchain. https://www.monad.xyz/(accessed August 13, 2025).
[51] P. Garamvolgyi, Y. Liu, D. Zhou, F. Long, M. Wu, Utilizing parallelism in smart contracts on decentralized blockchains by taming application-inherent conflicts, in: 2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE), 2022, pp. 2315–2326, https://doi.org/10.1145/3510003.3510086.
[52] P.S. Anjana, S. Kumari, S. Peri, S. Rathor, A. Somani, OptSmart: a space efficient optimistic concurrent execution of Smart contracts, Distrib. Parallel Databases 42 (2024) 245–297, https://doi.org/10.1007/s10619-022-07412-y.
[53] G. Mitenkov, I. Kabiljo, Z. Li, A. Spiegelman, S. Vusirikala, Z. Xiang, A. Zlateski, N. P. Lopes, R. Gelashvili, Deferred objects to enhance smart contract programming with optimistic parallel execution, (2024). https://doi.org/10.48550/arXiv.2405.06117.
[54] H. Qi, M. Xu, X. Cheng, W. Lyu, Latency-first smart contract: overclock the blockchain for a while, in: IEEE INFOCOM 2023 - IEEE Conference on Computer Communications, 2023, pp. 1–10, https://doi.org/10.1109/INFOCOM53939.2023.10228992.
[55] Y. Fang, Z. Zhou, S. Dai, J. Yang, H. Zhang, Y. Lu, PaVM: a parallel virtual machine for smart contract execution and validation, IEEE Trans. Parallel Distrib. Syst. 35 (2024) 186–202, https://doi.org/10.1109/TPDS.2023.3334208.
[56] Kindelia, GitHub. https://github.com/kindelia (accessed August 13, 2025).
[57] Fuel ignition is live! https://fuel.network/(accessed August 13, 2025).
[58] Linera | the real-time blockchain. https://linera.io/(accessed August 13, 2025).
[59] Sei. https://www.sei.io/(accessed August 13, 2025).
[60] eUTxO.Org - visual Blockchain Explorer for Cardano. https://eutxo.org/(accessed August 13, 2025).
[61] Bitcoinwiki. http://bitcoinwiki.org/wiki/utxo (accessed August 13, 2025).
[62] Machine learning, explained | MIT Sloan. https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained (accessed August 14, 2025).
[63] B. Yu, H. Zhao, T. Zhou, L. Chen, Y. Fan, TCHAO: parallelism with transaction classification and grouping by historical access objects to scale blockchain, J. King Saud Univ. Comput. Inf. Sci. 37 (2025) 67, https://doi.org/10.1007/s44443-025-00090-7.
[64] Proof of stake - bitcoin wiki. https://en.bitcoin.it/wiki/Proof_of_Stake (accessed August 14, 2025).
[65] M. Raikwar, N. Polyanskii, S. Müller, SoK: dAG-based consensus protocols, in: 2024 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), 2024, pp. 1–18, https://doi.org/10.1109/ICBC59979.2024.10634358.
[66] Canto.Io - layer-1 blockchain. https://canto.io (accessed August 14, 2025).
[67] C. Xu, C. Zhang, J. Xu, J. Pei, SlimChain: scaling blockchain transactions through off-chain storage and parallel processing, Proc. VLDB Endow. 14 (2021) 2314–2326, https://doi.org/10.14778/3476249.3476283.
[68] Verkle trees, Ethereum.Org. https://ethereum.org/en/roadmap/verkle-trees/(accessed August 14, 2025).
[69] initia-labs/movevm, (2025). https://github.com/initia-labs/movevm (accessed August 14, 2025).