Ethpar: Parallel Ethereum

Christina Peterson[1,2][0000−0002−8070−7633], Zachary Painter[1,3][0000−0001−8334−8237], and Victor Cook[1,4][0000−0002−9852−2581]

[1] Parallel Logic Corporation, USA
[2] `cpeterson@Ethpar.com`
[3] `zpainter@Ethpar.com`
[4] `vcook@Ethpar.com`

Abstract. A consensus mechanism is a mission-critical component of a blockchain, enabling the ability to securely append blocks to the blockchain such that all participants agree on block history. The essential properties of a blockchain network are security, scalability and decentralization. To scale network throughput and uphold decentralization, we present Parallel Ethereum (Ethpar), a hard fork of Ethereum that provides proof-of-stake consensus with support for parallel blocks. Ethpar leverages transaction commutativity to enable parallel blocks to be added to a slot alongside the beacon block. The transactions are organized in a memgraph, where transactions form edges that connect to each node corresponding to an address in its conflict list. Ethpar uses a deterministic conflict resolution scheme to partition the memgraph into commutative sub-components that does not require any additional communication beyond synchronizing the memgraph among parallel validators. To ensure that validators agree on the conflict lists for smart contracts, Ethpar employs static analysis of smart contract source code to produce the conflict lists without smart contract execution.

Keywords: Blockchain · Throughput · Commutativity · Consensus

1 Introduction

The engineering and business transformation accomplished in September 2022 (the "Merge") to migrate Ethereum from proof-of-work (PoW) to proof-of-stake (PoS) was nearly flawless in concept and execution. It was a grand experiment with many benefits. On the two year anniversary of the Merge, we recognize unintended consequences. PoS reduces electricity consumption [2], but cost and complexity have distanced the network from solo stakers towards pools, where financial resources are concentrated and yield is lower than a US Treasury bond (3.2% vs 4.8%). Staker frustration with low yield leads to risky hypothecation, detrimental to network security. A few of these pools make up more than half of the total stake, causing centralization.

The desire to maximize "tips" to proposers has created a centralized market for block building, where a few builders arbitrate the most profitable transactions. The centralization of digital currencies and their regulation has been explored [26,39], where fundamental vulnerabilities relating to centralization include the central authority being a single point of failure. The central authority may also favor certain participants while discriminating against other participants with respect to rules, penalties, and transaction processing. The builders must prioritize Maximal Extractable Value (MEV) to maximize profits, but this is vulnerable to corruption since front-running and arbitrage are commonly used to gain profits at the expense of honest participants.

The Merge itself did not address network performance, as neither throughput nor latency were improved in September 2022. The quest for better performance has spawned competitors to Ethereum, many of which support the Ethereum Virtual Machine (EVM) with the intention of luring away Ethereum developers and applications. Layer 1 is where the network earns fees and is most trusted by users, but instead of meeting the competition, Ethereum has favored Layer 2 and deprioritized Layer 1 solutions. Issues are difficult to address in an ecosystem that waits years for improvement proposals to be reviewed and implemented. In this paper, we present Parallel Ethereum (Ethpar), an unapologetic contribution to decentralize, increase throughput and align incentives to revitalize Layer 1.

Various Layer 1 and Layer 2 solutions have been introduced to address scalability. Sharding is a widely-accepted Layer 1 solution to scaling the blockchain while maintaining security and decentralization [8,40]. Sadly, cross-shard transactions require expensive synchronization protocols and communication between the consensus committees across different shards. This figured in the decision by Ethereum to extend the timeline for sharding years in the future.

The use of sidechains is a Layer 2 solution that enables large batch transactions to be processed offchain, while cryptographic proofs secure them to the main chain. However, Layer 1 is preferable because it provides a higher degree of decentralization and security than Layer 2 solutions. This preference is visible in the market as the great majority of stablecoin value is held on Layer 1 blockchains.

Our vision is to use parallel computing techniques to scale Layer 1 Ethereum while preserving the original philosophy of decentralization. We put this vision into practice with Parallel Ethereum (Ethpar), a blockchain that encompasses a hard fork of Ethereum to deliver proof-of-stake consensus with support for parallel blocks. Ethpar leverages the untapped power of redundant validators (i.e., validators not proposing a block) per slot by recruiting them as parallel validators. Each parallel validator proposes a block in parallel with the beacon block. Concurrency control for the parallel blocks is made possible by organizing the mempool transactions in a data structure referred to as a *memgraph*. The memgraph arranges transactions from the mempool into a graph structure where wallet or smart contract addresses are nodes and transactions are edges. An edge from address $a1$ to address $a2$ indicates a transaction that accesses both $a1$ and $a2$. Commutativity is a property such that two transactions are *commutative* if executing them in either order yields the same final state. As such, any edges in a connected subgraph represent transactions that are non-commutative.

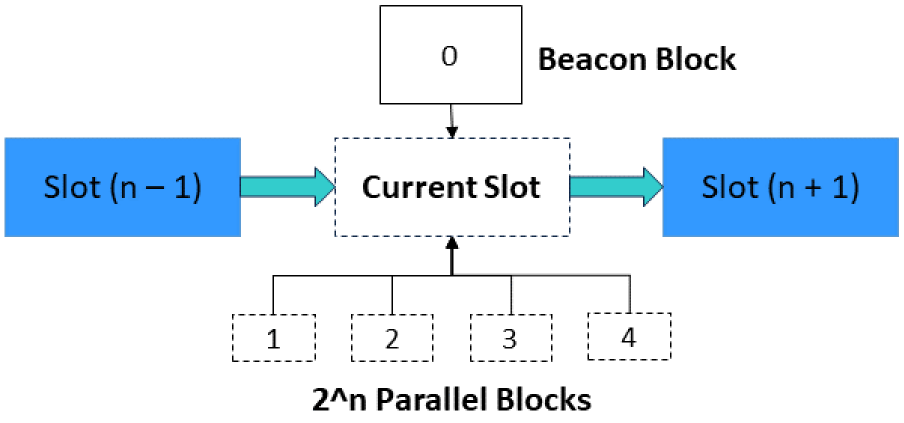


Fig. 1: Each slot contains 2^n commuting blocks, executed in parallel

We observe that two commutative blocks can be published with no defined ordering between them, so long as they both have a total ordering with any prior or subsequent blocks with which they do not commute. This produces a ledger where the computed state is deterministic, as all possible orderings for commuting blocks generate the same final output. Therefore, we devise an approach that enables validators to publish parallel blocks at each slot n, so long as their block commutes with every other block at slot n. To achieve this, we modify the Ethereum validator scheduling such that at each slot n, 2^m validators are selected, each with a unique numerical rank r. This is referred to as the "parallel committee" for slot n. Validators each propose a *single* block that commutes with every other block at slot n. We achieve this by proposing a deterministic conflict resolution scheme.

First, validators in a parallel committee directly peer with one another and fully synchronize their memgraphs. To ensure that each validator has the same view of the memgraph, we extend the functionality of the existing "seen" status for transactions, which is updated during transaction propagation in the execution client. A transaction can only be inserted into the memgraph if it has been seen by all selected validators. We call this new status "vetted" and track it using a data structure that includes all the validators in the parallel committee. This ensures that all validators in a parallel committee work with the same set of transactions.

Given a parallel committee of validators, each with an identical copy of the memgraph, a deterministic partitioning algorithm is used to find all groups of non-commuting transactions. Each transaction within a group must be assigned to the same block, to ensure commutativity between blocks within that slot. To achieve this, validators are initially assigned transactions from the memgraph using a deterministic function that maps transactions to a rank. This function is designed such that no two validators are assigned the same transaction, however, they may be assigned transactions within the same non-commuting group. To resolve this, we use a straightforward conflict resolution scheme where the validator with the lowest rank wins the entire set of non-commuting transactions. Since this scheme is inherently unfair, we tailored the deterministic function to map more transactions to higher ranks.

In this way, the Ethpar ledger becomes representative of a conflict-serializable precedence graph. Blocks in slot n have a total ordering with respect to blocks in slot $n+1$, and a partial ordering with respect to blocks in slot n. Any conflicting transaction at slot n will end up in the same block, ensuring that the order of conflicting transactions is the same in every possible execution. Validators utilize a fast and efficient conflict resolution scheme to divide commuting transactions based on their semantics. These conflicts are resolved before the block is built, giving validators freedom to construct their block so long as it satisfies the constraints of their agreed upon list of transactions. Our approach improves upon sharding based approaches, as all parallel blocks are stored within the same ledger, eliminating the need for costly two-phase locking schemes across shards. Ethpar is not mutually exclusive with sharding; it can be implemented in tandem with sharding to improve transaction throughput via parallel blocks at each shard. Additionally, our approach encourages decentralization because validators will ultimately obtain higher profits from building more parallel blocks rather than joining centralized entities that focus on building the most lucrative block possible.

2 Related Work

The demand for peer-to-peer networks that support acquisition or electronic transfer of digital currencies and assets without a trusted third party has accelerated over the past few decades. Ethereum [4], introduced by Buterin in 2014, delivers a blockchain with a built-in Turing-complete programming language that enables versatile smart contracts such as multi-stage options contracts or contracts with conditional dependencies on fluctuating prices. Ethereum originally employed proof-of-work as its consensus mechanism for reaching agreement among peers on the history of transactions. Although Ethereum was initially created to provide a superior platform for smart contracts, they later made a bold move to address one of the fundamental criticisms of proof-of-work - the enormous energy consumption required to achieve consensus. Ethereum changed from proof-of-work to proof-of-stake on September 15, 2022 known as “the Merge.” Instead of using network computing power as the mediator of good behavior, proof-of-stake requires a large amount of Ethereum’s digital currency Ether (ETH) to be staked by every peer serving as a validator. The most severe misbehavior penalty is the loss of the entire staked ETH, which outweighs the financial gains that could be achieved by malicious peers engaging in foul play on the network.

2.1 Trilemma

The blockchain trilemma, introduced by Buterin in 2017 [33], claims that no blockchain can simultaneously guarantee decentralization, security, and scalability. At best, a blockchain can guarantee two of these elements at the cost of sacrificing the third. Sharding is proposed as a solution to the trilemma [34], but research on sharding as a solution is still ongoing [22]. A robust decentralized blockchain permits a large number of participants to join the network. In this scenario, security is more vulnerable since the likelihood of a bad actor joining the network increases. Scalability is also hampered since the consensus mechanism takes more time to accommodate the large number of participants. Security ensures that the blockchain is tamper-resistant. Achieving security in a decentralized environment is challenging because there are no guarantees that the participants can be trusted. The solution for upholding security is to make the cost of misbehaving so high that it disincentivizes dishonest participation. Proof-of-work requires solving a cryptographic hash problem that is energy inefficient to append a new block to the blockchain, reducing scalability. Proof-of-stake requires the block proposer to hold a large monetary stake that is at risk of being lost upon misbehavior. While proof-of-stake overcomes some scalability issues, it threatens decentralization since the participants with the largest financial resources are the ones controlling the network. Scalability enables the blockchain to maintain a high transaction processing volume as network participation increases. The consensus mechanism is one of the primary bottlenecks to achieving scalability. Lowering the number of validators in proof-of-stake to reduce block time reduces decentralization, while lowering the energy cost of proof-of-work to speedup consensus weakens security. To ensure decentralization, nodes must be able to participate in the network with only moderate resources. Many highly scalable networks sacrifice decentralization because average users are unable to keep up with the CPU demand of processing significant transaction volume. Our solution mitigates this by partitioning the increased transaction throughput into blocks that are easy to execute in parallel. In Section 3 we analyze additional factors that affect decentralization, including staker pooling, and most severely, MEV.

2.2 Transaction Throughput

Many blockchains aim to address the scalability aspect of the blockchain trilemma by improving upon the low transaction throughput of initial blockchains. Bitcoin's average throughput is 8 transactions per second (TPS), while Ethereum's average throughput is 15 TPS [37]. Cardano [16] emerged in 2015 from peer reviewed blockchain research and obtains improved transaction throughput using Ouroboros [19] as its proof-of-stake protocol. Ouroboros uses a leader election process that randomly selects a leader with a probability that is proportional to their stake. Cardano achieves scalability through Ouroboros' ability to elect a quorum of consensus nodes for an epoch in a decentralized way. The blockchain itself is partitioned into shards, and a quorum of consensus nodes is elected concurrently for each shard. Cardano's maximum throughput is more than 1,000 TPS [3]. Ouroboros block finality is guaranteed after k blocks, where k is a network parameter set to 2160, which occurs in 12 hours or less. Cardano differs from our approach in its use of a Layer 2 solution for parallelism. Cardano achieves parallel transaction execution with hydra [5, 17], an off-chain protocol for quickly settling transactions which are then merged back into the main chain. In our approach, transactions are not settled off-chain. Instead, transactions are partitioned based on their conflicts so that they can be published in parallel blocks within the Layer 1 chain.

Solana [38], introduced by Yakovenko in 2018, incorporates proof-of-history consensus combined with proof-of-stake to reduce messaging overhead and achieves block finality in approximately 400 - 800 milliseconds. This is much faster than Bitcoin's 10 minute block finality and Ethereum's 12 second block finality [35]. Proof-of-history uses a cryptographically secure function to create a sequence of hashes where data can be timestamped into the sequence by appending the data into the state of the function, establishing an order among events. The verification of the history is *embarrassingly parallel* (i.e. a workload that can be split into parallel sub-workloads in a straightforward manner) because the sequence of hashes can be divided into slices and each slice verified in parallel on its own core. Solana's average throughput is 877 TPS, with a maximum theoretical throughput of 65,000 TPS [37].

Hedera Hashgraph [1], proposed in 2018 by Baird et al., aims to improve transaction throughput and security with the hashgraph consensus algorithm. The hashgraph consensus algorithm uses a gossip protocol combined with a timestamp for each transaction to determine its consensus order. Proof-of-stake is used to determine a node's influence on consensus, which is proportional to the amount of cryptocurrency that the node has staked. As the network grows, the nodes are divided into separate shards so that consensus for each shard can proceed in parallel. Hedera's average throughput is 1,544 TPS, with a maximum theoretical throughput of 10,000 TPS [37]. The time to reach block finality in Hedera Hashgraph is 3-5 seconds [15].

SEI [21], proposed in 2022 by Sei Labs, uses parallelism in several aspects of their design to reduce transaction latency and improve throughput. SEI uses a Twin-Turbo consensus that begins optimistic block processing immediately after the block is proposed, which runs concurrently with the prevote and precommit steps. If the block is accepted, the data written to the cache during optimistic block processing will be committed. If the block is rejected, the data in the cache is discarded and future rounds for this block height will not use optimistic block processing. Transactions are processed in parallel by mapping transaction messages to the keys they need to access in the key-value store. Messages that update different keys may be run in parallel. Dependencies between transactions are determined through a Directed Acyclic Graph (DAG) of dependencies based on the resources that each transaction needs to access. SEI's throughput is approximately 12,500 TPS, and reaches block finality in 380 milliseconds [27]. This approach differs from Ethpar in that it only enables parallelization of transactions within the same block, and does not enable multiple blocks to be published or executed concurrently.

Nightshade [29], originally proposed in 2019, upgraded to version 2.0 in 2024 which incorporates some of the latest advancements in zero-knowledge proofs. Block producers and validators in Nightshade build a single blockchain referred to as the main chain. Sharding is used to split the state of the main chain into n shards. Nightshade focused their approach on state sharding since data storage for the state grows over time even if the transactions per second remains the same. The addition of stateless validation improves per-shard throughput in Nightshade by enabling nodes to hold the state in memory.

Ethereum undertook another improvement when the Dencun upgrade was released on Mainnet on March 13, 2024 [24]. Dencun introduces proto-danksharding, a technique that benefits Layer 2 solutions by enabling Ethereum to store large transaction data off-chain. Data blobs, which replace the transaction calldata, enhance the disc space usage for validator nodes because transaction calldata must be retained by the nodes forever and blob data can be pruned after two weeks. Proto-danksharding is a stepping stone towards full danksharding. A danksharding system will divide the Ethereum blockchain into shards, allowing for the parallel execution of transactions in separate shards.

2.3 Multiple Block Proposers Per Slot - A Solution For Censorship

Any blockchain that enables free choice of transaction selection during block building is vulnerable to *censorship* - the deliberate exclusion of a transaction from a block. An adversary may be incentivized to prevent a transaction from being included in a block if it will result in profit loss. In this case, the adversary may need to bribe the block proposer with an amount larger than the fee of the transaction to be censored to ensure that it is not included in a block. Neuder et al. [23] recognize that the cost of censorship increases linearly with the number of block proposers that have the option to include the transaction. The authors present the concept of multiple proposers per slot. Each proposer independently builds a payload, where identical transactions within each payload are permitted. Fees for the doubly included transactions are divided among the proposers. The payloads are concatenated together to form a single block. The authors outline the various outcomes of the concatenated payloads and prioritize the cases when assigning proposer boost for resolving a chain split.

Fox et al. [13] investigate censorship resistance in time sensitive auctions hosted by proof-of-stake consensus. The block proposer receives the bids for blocks of transactions with a tip for the proposer if the transactions are included in the proposed block. The authors consider two designs. The first design conducts the auction over multiple slots with a different proposer for each slot. The analysis of this scenario shows sufficient censorship resistance when the number of blocks is larger than the number of bidders, which is undesirable under the common case of requiring a short auction window. The second design is to have multiple concurrent block proposers where a colluding bidder must bribe multiple proposers to successfully censor a transaction. This yields improved censorship resistance because the cost of censoring a transaction increases linearly with the number of block proposers. The innovations by Neuder et al. [23] and Fox et al. [13] are similar to Ethpar in that they use multiple block proposers per slot. The distinguishing feature of Ethpar is that the block payload for each proposer contains unique transactions that commute with the transactions in other payloads, enabling safe parallel execution that increases transaction throughput.

3 PoS and MEV

Maximal Extractable Value (MEV) is maximum obtainable profit that can be achieved through a valid ordering of a subset of the pending transactions in the block [20]. MEV previously was an acronym for Miner Extractable Value when proof-of-work was prevalent, but changed to Maximal Extractable Value when proof-of-stake and other consensus mechanisms became mainstream. Searchers use bots to scan the blockchain transactions for MEV opportunities and submit a profitable transaction with a high gas fee to the network when a sequence of operations capitalizing on MEV is found. Decentralized Exchange (DEX) arbitrage, where a token is purchased on one exchange and immediately sold at a higher price on a different exchange due to differing exchange prices, is a highly sought after MEV opportunity. Front-running is an MEV opportunity where a profitable transaction is detected and replicated with the frontrunner's address and a higher gas fee to obtain the MEV away from the original searcher. The unethical practice of front-running has motivated solutions such as the Hash-Mark-Set [6] algorithm, which associates a sequence order for each transaction that prevents the strategic placement of a transaction ahead of another transaction for profit gains. A sandwich trade is another MEV opportunity where a searcher uses a bot to scan the blockchain transactions for a large DEX trade that is likely to increase the price of a trading pair. A searcher can then issue a buy order before the large trade, and a corresponding sell order after the trade to make a profit.

Ethereum's move to proof-of-stake consensus has endured a greater negative impact from MEV because the resulting effect is validator centralization. The large ETH stake required to be a validator is an incentive for the average user to join a staking pool to lower the costs to activate a set of validator keys. As the staking pools grow larger, their MEV extraction capabilities improve, leaving very little MEV on the table for solo validators to capitalize on. This further incentivizes a user to join a large staking pool, leading to centralization among the pool of validators. Proposer-Builder Separation is intended to mitigate the impact of MEV by removing MEV extraction from validators and giving it to new entities known as block builders that order transactions and build blocks. MEV-Boost [36], developed by Flashbots, is an implementation of Proposer-Builder Separation developed for Ethereum's proof-of-stake. The block builders create a transaction bundle that is blinded and has an associated fee. The blind bundle is placed in an auction to be included in the beacon block. The validator chooses the bundle with the highest fees and the block builder publishes the full block body upon receiving the signed block proposal. Although the Proposer-Builder Separation successfully took the block building and potentially corrupt MEV opportunities away from the large validator staking pools, it reassigned this task to the block builders. MEV extraction is still a centralizing force among block builders since validators are blindly signing off on the block proposal with the highest fee without concern for how this fee was obtained.

4 Motivation

The trend towards enhancing blockchain transaction throughput has incorporated parallelism in some aspect of consensus or block construction. Solana divides the sequence of hashes forming the proof-of-history into slices for parallel processing. While proof-of-history vastly improves the time and energy efficiency of consensus in comparison to proof-of-work, it doesn't address the scalability concerns for block building. Cardano, Hedera Hashgraph, and full danksharding divide the nodes into shards, enabling consensus and block processing to proceed concurrently for each shard. The drawback of sharding is that a transaction that spans multiple shards requires expensive synchronization protocols such as two-phase locking to ensure atomicity and isolation for its operations [8].

SEI uses a DAG to identify transaction dependencies and executes non-conflicting transactions in parallel within a block. The parallelism achievable in SEI's block processing is limited by block size. Although the block size is a configurable parameter, increasing the block size results in increased bandwidth and increased latency to propagate the block [9].

Our goal is to enhance block construction by enabling entire blocks to be built in parallel, referred to as Parallel Ethereum (Ethpar). We use a similar technique as SEI regarding the parallel execution of commutative transactions, i.e. transactions with disjoint dependencies. The distinguishing feature of our approach is that the use of parallel blocks has much better scalability potential since block size is not a limiting factor for the number of transactions that can be processed in parallel. A subset of validator nodes are randomly selected to serve as a parallel validator for validation of the parallel blocks. Ethpar mitigates the corruption involved in MEV extraction because the addition of the parallel validator role incentivizes validators to prioritize transaction commutativity when building blocks rather than MEV.

5 Parallel Ethereum (Ethpar)

The core of Parallel Etherem (Ethpar) is the memgraph. In the memgraph, ledger addresses are represented as nodes, with transactions forming undirected edges between any address they read or write, shown in Figure 2a. Intuitively, this graph can quickly be partitioned into commutative sets of transactions by selecting all unconnected components. The construction of the memgraph is possible because all transaction logic is known in advance of its execution, including the storage variables and addresses the transaction may access.

Validators require knowledge of a transaction's conflicts to insert it into the memgraph. Furthermore, the conflicts must be accurate with respect to the state of the ledger, as validators working in parallel will be unaware of state changes made by each other during the block interval. To prevent state-changes from creating unexpected transaction conflicts for validators working in parallel, we define a transaction's *conflict list*:

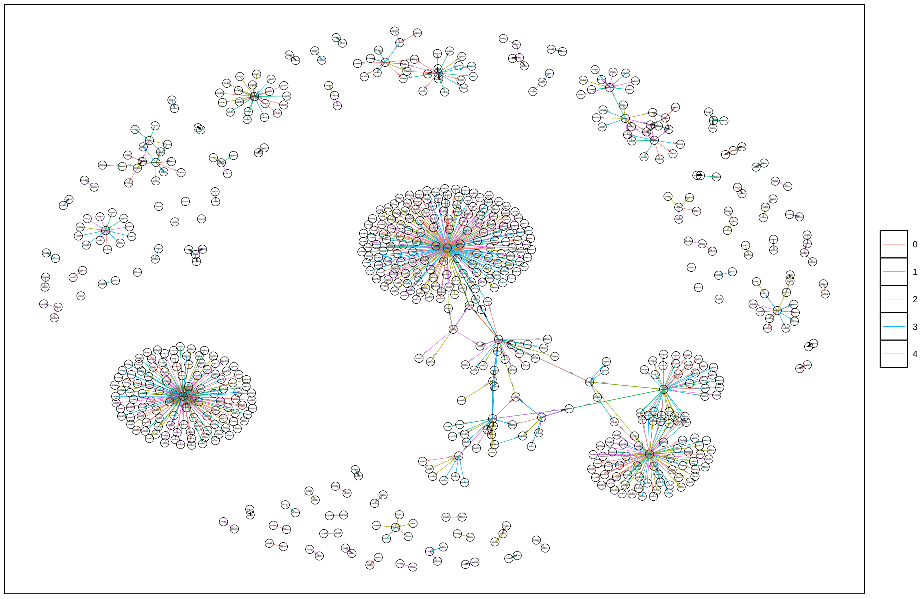
Definition 1. *A transaction t's **conflict list** is the set of all world state elements that may be read or written during the execution of t.*

A transaction's conflict list helps identify transactions that commute, regardless of the present, or future world state. In Ethereum, the *world state* refers to the state of all accounts, each of which has an associated ether balance, and storage [11].

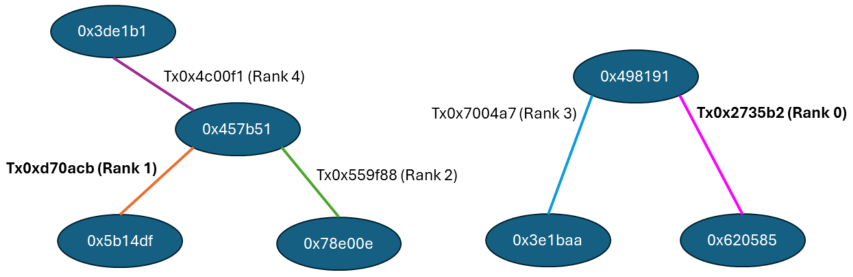
Array accesses commute if they operate on unique indices, for example, writes to $array[x]$ and $array[y]$ commute if $x \neq y$. For this reason, each index of an indexable storage object is treated as a unique world state element. This provides finer granularity in identifying transaction conflicts. Local variables are not included in the conflict list unless they are used to index a storage object. For example, the conflict list for the statement $array[x] = y$ would be {array[x]}, if x and y are local variables. Validators use the conflict list when inserting transactions into the memgraph to ensure that the edges in the memgraph will not change in response to state changes in the ledger. We explore a detailed case related to this in Section 5.4, using the TetherToken and Uniswap smart contracts.

Our definition for a transaction's conflict list is inspired by EIP-2930 [10], which introduced *access lists*. Access lists were introduced to optimize storage accesses by including a list of storage keys that will be accessed by a transaction during execution. However, the access list for a smart contract is dependent upon the state of the ledger at the time of execution. For this reason, it is not sufficient to represent all possible conflicts between transactions in parallel blocks. Furthermore, computing the access list for a transaction requires the transaction to be executed within a sand-boxed EVM. A transaction's conflict list can be computed more efficiently than its access list via static analysis of a smart contract's abstract syntax tree.

In Ethpar, A subset of validators are selected as a *parallel validator committee* at each slot (Figure 3). Parallel validators cooperate to append 2^m commuting blocks at their assigned slot n. Each parallel validator is assigned a unique rank $r \in \{1, 2^m - 1\}$ in slot n. Rank 0 is reserved for the beacon block. Once a validator is selected as a parallel validator, it must open a peer-to-peer connection in the execution client with the other parallel validators. By directly peering with each other, parallel validators can construct a memgraph comprising transactions that are guaranteed to be seen by the other parallel validators. This ensures that



(a) Memgraph of Holesky network.



(b) Memgraph snippet. Each transaction is represented as an edge connecting the addresses that it accesses. Each transactions is also assigned a validator rank. The validator with the lowest rank wins all transactions in a connected component, which represents a non-commutative set of transactions.

Fig. 2: Memgraph Overview.

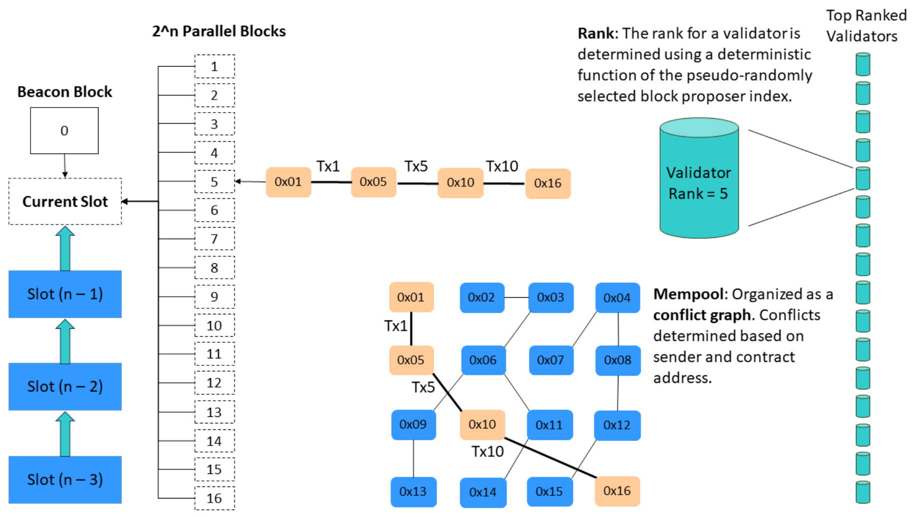


Fig. 3: Parallel blocks overview. Each slot supports 2^n commuting blocks, appended in parallel by a parallel validator committee.

parallel validators have a consistent view of the memgraph, enabling them to divide transactions amongst themselves without additional communication beyond that of synchronizing their memgraphs. Once validators have arrived at the same view of the memgraph, they execute a deterministic algorithm to partition the memgraph, ensuring that each validator receives a block of transactions that is entirely commutative with every other validators' block. If peer $p1$ and peer $p2$ are selected as validators for a particular slot, and $p1$ observes that $p2$ has an inconsistent view of the memgraph, then $p1$ will recuse itself from block production for the corresponding slot. This action protects the integrity of the block produced by $p2$ in the circumstance that $p2$ did not observe an inconsistency and proceeded with block production.

5.1 Managing a Transaction's Vetted Status

In order for a parallel committee to partition the memgraph into commutative sets, they must ensure that their respective views of the memgraph are identical. To achieve this, we define the *seen* and *vetted* statuses for a transaction.

Definition 2. *A transaction t is* ***seen*** *by peer p from the perspective of peer p_0 if p propagates t to p_0.*

Definition 3. *A transaction t is* ***vetted*** *by a parallel committee from the perspective of peer p_0 if it has been* ***seen*** *by each validator in the parallel committee from the perspective of peer p_0.*

A transaction is only inserted into the memgraph once it has obtained the vetted status. This enables a shared view of the memgraph, which is necessary for a deterministic conflict resolution that does not require any additional communication to partition the memgraph.

The seen status for a transaction is updated during the execution client's transaction propagation to other peers. A transaction $t1$ is guaranteed to have been seen by peer $p1$ if a peer receives $t1$ from $p1$. For this reason, a peer $p2$ marks $t1$ as "received" from $p1$ upon receiving the message of $t1$ by $p1$. Since peer $p1$ needs confirmation that peer $p2$ received transaction $t1$, peer $p2$ must send transaction $t1$ back to peer $p1$ to serve as an acknowledgement of receiving transaction $t1$. Peer $p2$ marks transaction $t1$ as "sent" to peer $p1$ upon sending $t1$ back to $p1$. The acknowledgement message ultimately doubles the number of messages for transaction propagation by the execution client. However, since the number of parallel validators is only a small subset of the total number of validators the amount of additional messages generated does not cause congestion in the execution client communication network.

Peer $p2$ considers transaction $t1$ as seen by peer $p1$ if 1) $p1$ has sent $t1$ to $p2$, and 2) $p2$ has has sent $t1$ back to $p1$. The transaction propagation of $t1$ from $p2$ to $p1$ reaches a stopping condition if both of these criteria are satisfied. The seen transaction status for each peer is maintained in a hashmap, where the peer is the key and the set of seen transactions is the value. Prior to transaction selection, each peer consults this hashmap to determine which transactions have been vetted (i.e. seen by all parallel validators). The vetted transactions are inserted into the memgraph, where the transactions will then be partitioned for the parallel validators transaction assignment.

5.2 Deterministic Conflict Resolution for Memgraph Commutative Partitions

The transactions in the memgraph must be partitioned into N ranks (N equals the number of parallel validators plus one for the beacon block validator). The goal is to partition the transactions such that the transactions in each partition are commutative with transactions in other partitions. Partitioning the memgraph for the parallel validators is achieved by using a breadth-first search on the memgraph to find all connected components. Transactions within a connected component are non-commutative with each other, but commute with transactions in different connected components. A simple solution is to distribute the connected components to the parallel validators such that they are assigned all transactions from the connected component. The challenge is enabling the parallel validators to know which connected component they are assigned without communication. We achieve this by computing the $UnsignedBigInteger$ value for the transaction hash and applying a modulo operation to distribute the transactions into a specified number buckets that is greater than or equal to the number of parallel validators plus one for the beacon block. Each parallel validator is assigned to one or more buckets. A set of parallel validators contending for each connected component is determined based on the assigned buckets of the transactions for the connected component. The lowest rank validator from the set of parallel validators wins the connected component.

Figure 2b shows an example of the memgraph with a view of five transactions. Each transaction is connected to the the addresses in its access list. The rank representing validator assignment for each transaction is listed next to its transaction label. The transactions in a connected component are non-commutative with each other and commutative with transactions in separate components. The validator with the lowest rank wins the transactions in the connected component, expressed in boldface print.

Although this strategy is deterministic and requires no communication among the parallel validators, it is unfair. We incorporate two strategies to mitigate unfairness. First, the top N connected components based on number of transactions are distributed to the N validators such that each validator is only assigned one of the top N connected components to improve load balancing. The connected components are sorted in descending order based on number of transactions, where ties are broken by a function of the connected component's transaction hashes. Rank 0 through rank $N-1$ are assigned one of the top N connected components based on the sorted order.

Second, lower ranks are assigned more buckets. The number of buckets is set to 2^{N+1}, where N is the number of ranks (beacon block validator plus the parallel validators). The first two buckets (bucket 0 and bucket 1) are assigned to rank 0. Each subsequent bucket i is computed using Equation 1.

$$rank = \lfloor \log_2 i \rfloor - 1, i \geq 2 \tag{1}$$

For example, if there are two parallel validators, there are three ranks and $2^4 = 16$ buckets. Rank 0 is assigned bucket 0 through bucket 3. Rank 1 is assigned bucket 4 through bucket 7. Rank 2 is assigned bucket 8 through bucket 15. Rank 2 is initially assigned more transactions because it is most likely to relinquish transactions when contending with another parallel validator for a connected component. After the validator has determined its assigned transactions, it proceeds with the block building process using only transactions that it has been assigned.

5.3 Parallel Block Execution

The memgraph naturally partitions pending transactions into commutative sets, which are deterministically assigned to parallel validators. These commutative sets lend themselves to concurrent execution. Although EVM support for concurrent execution is not widespread, multiple EVM instances can be instantiated in parallel to process the blocks created by each parallel validator committee.

Figure 4 gives an overview of parallel block execution. Each parallel EVM instance is instantiated from a copy of the world-state at the start of the slot, and executes a single block. After each instance is finished, the resulting world-states are sequentially merged. If two world-states contain updates to overlapping storage variables, it means that their corresponding blocks are not actually commutative, and one must be rejected. The block rejection process is further described in Section 6.1.

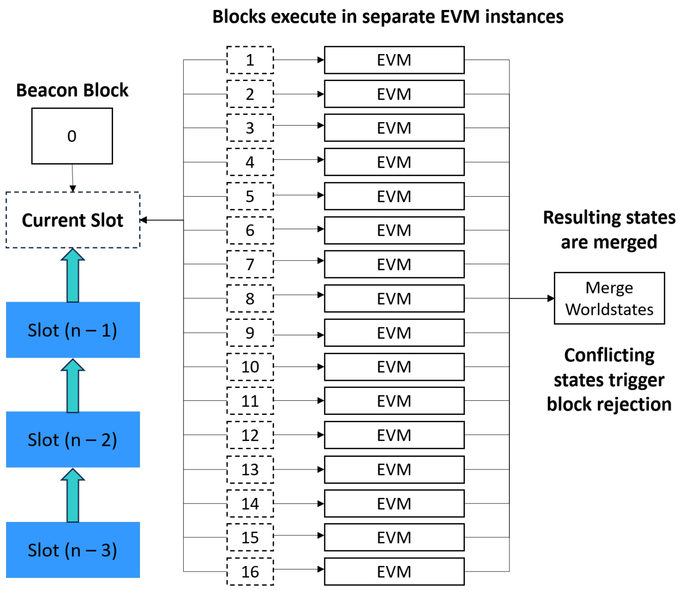

Blocks execute in separate EVM instances
Beacon Block
0
Current Slot
Slot (n – 1)
Slot (n – 2)
Slot (n – 3)
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
EVM
Resulting states are merged
Merge Worldstates
Conflicting states trigger block rejection

Fig. 4: Parallel block execution

In this approach, each EVM acts as a concurrent process, writing changes to a copy of shared data without regard for synchronization with other concurrent processes. Afterward, those changes are committed back to the shared database. This is similar to some Software Transaction Memory (STM) [28] or Optimistic Concurrency Control (OCC) [14] implementations, which are well studied for their benefit on multicore hardware when contention is low. Our approach also utilizes graph-based partitioning to assign conflicting transactions to the same block, in order to execute them sequentially. Similar to traditional implementations [7, 18, 25], the intuition is that conflicting transactions should be executed sequentially by a single process, while non-conflicting transactions are executed in parallel, with little synchronization overhead. As such, our approach is as a novel application of state-of-the-art concurrency techniques to decentralized networks. OCC is an optimistic algorithm, meaning processes perform their operations under the assumption that no concurrent processes will perform any conflicting operations. In our approach, we nearly guarantee this assumption will hold true, as our memgraph-based consensus algorithm always distributes groups of conflicting transactions to the same validator rank, and therefore the same block.

Load balancing is a concern if transactions in the mempool frequently conflict. In this scenario, there would be a low number of unconnected subcomponents in the memgraph, limitting the number of transactions allocated to certain validator ranks. However, this is not common in the average use case. Eth transfers are commutative if they operate on different addresses, and in Ethereum, there are hundreds of thousands of wallet addresses active daily [12]. Additionally, due to the gas-based execution system in Ethereum, smart contracts are typically designed to contain simple logic that uses as little gas as possible. These contracts can often be parallelized, as analyzed in section 5.4.

Our approach yields a ledger that is highly parallel. In a decentralized ledger, each block will be executed thousands of times as new nodes synchronize themselves with the network. By computing the memgraph at each slot, a small committee of parallel validators perform a powerful transaction partitioning step that ensures the efficient execution of transactions by all nodes for the lifetime of the ledger.

5.4 Static Analysis of Smart Contract Conflicts

Conflicts between smart contract calls can be detected using their conflict lists. The ETH_CREATEACCESSLIST JSON-RPC method can be used to generate the access list for a smart contract call, which can then be used to produce the conflict list, but this requires the transaction to be executed in full. In an effort to avoid this additional computation, we employ static analysis of smart contract source code to produce conflict lists without executing them. As an example, we explore a case study of the TetherToken smart contract [31], and Uniswap [32].

Listing 1.1 gives the TRANSFER method of the TetherToken smart contract. The method takes two parameters as input, *_to* and *_value*. On lines 13 and 14, the state variable *balances* is updated at indices corresponding to *msg.sender* and *_to*. Furthermore, the balance of *owner* is updated conditionally on line 16. Since branching is only evaluated at runtime, the *conflict list* includes any world state element that is read/written anywhere in the contract, regardless of branching.

In this example, the conflict list for TRANSFER(_to, _value) is {balances[msg.sender], balances[_to], balances[owner], owner}. Note that by reading the storage variable *owner* on line 16, this code creates a single point of contention about which all calls to TRANSFER will conflict. This type of problem can be resolved using traditional strategies for concurrent programming, such as by treating *balances*[*owner*] as a thread-local accumulator. However, as it stands, calls to TRANSFER cannot commute. This does not necessarily prevent TRANSFER calls from commuting with other methods within the TetherToken smart contract.

Listing 1.2 gives the APPROVE method of the TetherToken smart contract. This method only makes changes to the shared *allowed* map, using *msg.sender* and *_spender* as keys. Both of these keys are local to the method call, and will therefore be known to all parallel validators. As a result, the conflict list for APPROVE(_spender, _value) is simply {allowed[msg.sender][_spender]}. This enables a high degree of parallelism between calls to APPROVE, as any two calls

```
address public owner;
mapping(address => uint) public balances;
event Transfer(
    address indexed from, address indexed to, uint value);
...
function transfer(address _to, uint _value)
        public onlyPayloadSize(2 * 32) {
    uint fee = (_value.mul(basisPointsRate)).div(10000);
    if (fee > maximumFee) {
        fee = maximumFee;
    }
    uint sendAmount = _value.sub(fee);
    balances[msg.sender]= balances[msg.sender].sub(_value);
    balances[_to] = balances[_to].add(sendAmount);
    if (fee > 0) {
        balances[owner] = balances[owner].add(fee);
        Transfer(msg.sender, owner, fee);
    }
    Transfer(msg.sender, _to, sendAmount);
}
```

Listing 1.1: TetherToken transfer method

```
mapping (address=> mapping (address=> uint)) public allowed;
...
function approve(address _spender, uint _value)
        public onlyPayloadSize(2 * 32) {
    require(!((_value != 0)
        && (allowed[msg.sender][_spender] != 0)));
    allowed[msg.sender][_spender] = _value;
    Approval(msg.sender, _spender, _value);
}
```

Listing 1.2: TetherToken approve method

```
1 mapping(address => uint) public balanceOf;
2 ...
3 function transfer(address to, uint value) private {
4     balanceOf[msg.sender] = balanceOf[msg.sender].sub(value);
5     balanceOf[to] = balanceOf[to].add(value);
6     emit Transfer(from, to, value);
7 }
```

Listing 1.3: Uniswap transfer method

with a unique *msg.sender* and *_spender* will commute. Furthermore, calls to APPROVE can be executed in parallel with calls to TRANSFER so long as they originate from unique *msg.sender* addresses.

Listing 1.3 gives the TRANSFER method of the Uniswap smart contract. The method takes as input, *msg.sender*, *to*, and *value*. On lines 4 and 5, the state variable *balanceOf* is updated at indices corresponding to *msg.sender* and *to*. Unlike TetherToken, this contract does not have a single point of contention. Only *msg.sender* and *to* are used as keys, both of which are local. The conflict list for this method would be {balanceOf[msg.sender], balanceOf[to]}. Any two Uniswap TRANSFER calls can be executed in parallel so long as they have different *msg.sender* and *to* fields.

In cases like TetherToken and Uniswap, the conflict list will closely resemble the average-case access list. Validators can save substantial amounts of computation time by retrieving conflict lists through static analysis wherever possible, rather than through the ETH_CREATEACCESSLIST JSON-RPC method.

6 Protocol Enforcement

The Ethpar protocol relies on the commutativity of blocks that occupy a shared slot. Due to the presence of inter-transaction conflicts, validators sharing a slot are partially restricted in their freedom to select transactions from the mempool. This is necessary to preserve the commutativity of all blocks published by the parallel committee at a slot.

Like all PoS blockchains, Ethpar enforces the protocol by punishing violators with financial penalties. In Ethereum, meting out these penalties is called "slashing" [30]. The maximum penalty for an individual validator is to lose the entire stake, currently 32 ethers. Rewards and penalties are described in detail in Ethereum's documentation [30].

In addition to those already implemented in Ethereum, Ethpar has two additional actions that are subject to penalty for violations of the parallel block commutativity rules. These are (a) Boosting - a violation due to including a transaction in the block that was not in the memgraph, and (b) Censorship - a violation where the builder excludes a transaction that should have been in the block according to the memgraph of vetted transactions.

6.1 Boosting

In an effort to collect more fees, a validator can select their transactions from a version of the memgraph that is not fully synchronized with their parallel committee. This is called *boosting* and is a violation of the Ethpar protocol. Boosting can lead to unforeseen conflicts between the blocks published by that committee. To prevent this occurrence, validators include a *memgraph hash* field in their published block, which contains the hash of the memgraph after all transactions were vetted for the current slot. All honest validators in a parallel committee will publish blocks with matching *memgraph hash* fields so long as their memgraphs are equivalent, as described in Section 5.1. The memgraph at slot n can be reconstructed by inserting each transaction from each block at slot n into a freshly initialized memgraph. In the case that a validator produces a conflict through malicious behavior, the network will be able to identify which validator is at fault by checking the assigned rank of each transaction in the reconstructed memgraph for slot n.

The penalty for publishing a block that does not commute with the majority of blocks at its slot is slashing. The non-commuting block is removed and the transactions within return to the mempool.

6.2 Censorship

It is generally agreed that censorship of transactions at the network level is undesirable [13, 23]. Solving the censorship problem on Ethereum is a current research topic, as discussed in Section 2.3. The practice of transaction censorship is a threat to decentralization because it enables an adversary to control which transactions to include in blocks. The detection of censorship is performed in a similar manner as described in Section 6.1, where the memgraph hash is checked for equivalence with the other parallel validators. If the equivalence check passes, the memgraph is reconstructed and the deterministic conflict resolution algorithm of Section 5.2 is applied to determine if each block contains the expected transactions based on the associated rank for slot n.

7 Conclusion

Ethpar leverages transaction commutativity to deliver enhanced block construction through parallel blocks. The deterministic conflict resolution scheme for handling conflicts among transaction assignments per block is made possible through a synchronized memgraph that contains only transactions with the vetted status (i.e. transactions that have been seen by all parallel validators). Each parallel validator partitions the memgraph using the deterministic conflict resolution scheme and selects transactions for their block based on their assigned rank. The identification of conflicts in the memgraph is dependent on knowing which addresses a transaction accesses. To account for state variables in a smart contract, Ethpar uses static analysis of smart contract source code to efficiently produce conflict lists without smart contract execution.

References

1. Baird, L., Harmon, M., Madsen, P.: Hedera: A public hashgraph network & governing council. White Paper **1**(1), 9–10 (2019)
2. Bentov, I., Lee, C., Mizrahi, A., Rosenfeld, M.: Proof of activity: Extending bitcoin's proof of work via proof of stake [extended abstract]. ACM SIGMETRICS Performance Evaluation Review **42**(3), 34–37 (2014)
3. Bhalla, A.: Top cryptocurrencies with their high transaction speeds. https://www.blockchain-council.org/cryptocurrency/top-cryptocurrencies-with-their-high-transaction-speeds/, accessed: 2024-06-07
4. Buterin, V., et al.: A next-generation smart contract and decentralized application platform. white paper **3**(37), 2–1 (2014)
5. Chakravarty, M.M., Coretti, S., Fitzi, M., Gaži, P., Kant, P., Kiayias, A., Russell, A.: Fast isomorphic state channels. In: Financial Cryptography and Data Security: 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part II 25. pp. 339–358. Springer (2021)
6. Cook, V., Painter, Z., Peterson, C., Dechev, D.: Read-uncommitted transactions for smart contract performance. In: 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS). pp. 1960–1970. IEEE (2019)
7. Curino, C., Jones, E.P.C., Zhang, Y., Madden, S.R.: Schism: a workload-driven approach to database replication and partitioning. Very Large Data Base Endowment Inc.(VLDB Endowment) (2010)
8. Dang, H., Dinh, T.T.A., Loghin, D., Chang, E.C., Lin, Q., Ooi, B.C.: Towards scaling blockchain systems via sharding. In: Proceedings of the 2019 international conference on management of data. pp. 123–140 (2019)
9. Default for 'blockparams.maxbytes' consensus parameter may increase block times and affect consensus participation. https://github.com/cometbft/cometbft/security/advisories/GHSA-hq58-p9mv-338c, accessed: 2024-06-10
10. Eip-2930: Optional access lists. https://eips.ethereum.org/EIPS/eip-2930, accessed: 2024-06-26
11. Ethereum accounts. https://ethereum.org/en/whitepaper/#ethereum-accounts, accessed: 2024-9-16
12. Ethereum daily active addresses. https://ycharts.com/indicators/ethereum_daily_active_addresses, accessed: 2024-09-11
13. Fox, E., Pai, M., Resnick, M.: Censorship resistance in on-chain auctions. arXiv preprint arXiv:2301.13321 (2023)
14. Härder, T.: Observations on optimistic concurrency control schemes. Information Systems **9**(2), 111–120 (1984)
15. Hbar. https://hedera.com/hbar, accessed: 2024-06-07
16. Hoskinson, C.: Why we are building cardano. https://whitepaper.io/document/581/cardano-whitepaper, accessed: 2024-06-07
17. Jourenko, M., Larangeira, M., Tanaka, K.: Interhead hydra: Two heads are better than one. In: The International Conference on Mathematical Research for Blockchain Economy. pp. 187–212. Springer (2022)
18. Kallman, R., Kimura, H., Natkins, J., Pavlo, A., Rasin, A., Zdonik, S., Jones, E.P., Madden, S., Stonebraker, M., Zhang, Y., et al.: H-store: a high-performance, distributed main memory transaction processing system. Proceedings of the VLDB Endowment **1**(2), 1496–1499 (2008)

19. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: A provably secure proof-of-stake blockchain protocol. In: Annual international cryptology conference. pp. 357–388. Springer (2017)
20. Kjaerstad, M.: Maximal extractable value (mev). https://ethereum.org/en/developers/docs/mev/, accessed: 2024-06-10
21. Labs, S.: Sei: The layer 1 for trading. https://github.com/sei-protocol/sei-chain/blob/main/whitepaper/Sei_Whitepaper.pdf, accessed: 2024-06-07
22. Liu, Y., Liu, J., Salles, M.A.V., Zhang, Z., Li, T., Hu, B., Henglein, F., Lu, R.: Building blocks of sharding blockchain systems: Concepts, approaches, and open problems. Computer Science Review **46**, 100513 (2022)
23. Neuder, M., Resnick, M.: Concurrent block proposers in ethereum. https://ethresear.ch/t/concurrent-block-proposers-in-ethereum/18777/1, accessed: 2024-09-05
24. Patairya, D.K.: What is the ethereum dencun upgrade, and why is it important? https://cointelegraph.com/explained/what-is-the-ethereum-dencun-upgrade-and-why-is-it-important, accessed: 2024-06-10
25. Prasaad, G., Cheung, A., Suciu, D.: Handling highly contended oltp workloads using fast dynamic partitioning. In: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. pp. 527–542 (2020)
26. Raskin, M., Yermack, D.: Digital currencies, decentralized ledgers and the future of central banking. In: Research handbook on central banking, pp. 474–486. Edward Elgar Publishing (2018)
27. Sei. https://www.sei.io/, accessed: 2024-06-07
28. Shavit, N., Touitou, D.: Software transactional memory. In: Proceedings of the fourteenth annual ACM symposium on Principles of distributed computing. pp. 204–213 (1995)
29. Skidanov, A., Polosukhin, I., Wang, B.: Nightshade: Near protocol sharding design 2.0. https://discovery-domain.org/papers/nightshade.pdf, accessed: 2024-09-12
30. Smith, C.: Proof-of-stake rewards and penalties. https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/rewards-and-penalties/, accessed: 2024-09-06
31. Tethertoken smart contract, etherscan. https://etherscan.io/address/0xdac17f958d2ee523a2206206994597c13d831ec7#code, accessed: 2024-08-13
32. Uniswap smart contract, github. https://github.com/Uniswap/v2-core, accessed: 2024-08-15
33. Sharding faq. https://vitalik.eth.limo/general/2017/12/31/sharding_faq.html, accessed: 2024-09-11
34. Why sharding is great: demystifying the technical properties. https://vitalik.eth.limo/general/2021/04/07/sharding.html, accessed: 2024-09-11
35. What is block time in blockchain? https://www.nervos.org/knowledge-base/block_time_in_blockchain_(explainCKBot), accessed: 2024-06-07
36. What is mev-boost? https://docs.flashbots.net/flashbots-mev-boost/introduction, accessed: 2024-06-10
37. What is transactions per second (tps)? https://chainspect.app/blog/transactions-per-second-tps, accessed: 2024-06-07
38. Yakovenko, A.: Solana: A new architecture for a high performance blockchain v0.8.13. Whitepaper (2018)
39. Yermack, D.: Corporate governance and blockchains. Review of finance **21**(1), 7–31 (2017)

40. Zamani, M., Movahedi, M., Raykova, M.: Rapidchain: Scaling blockchain via full sharding. In: Proceedings of the 2018 ACM SIGSAC conference on computer and communications security. pp. 931–948 (2018)