DeFi and NFTs Hinder Blockchain Scalability

Lioba Heimbach, Quentin Kneip, Yann Vonlanthen, Roger Wattenhofer

ETH Zurich
Zurich, Switzerland

Abstract. Many classical blockchains are known to have an embarrassingly low transaction throughput, down to Bitcoin's notorious seven transactions per second limit. Various proposals and implementations for increasing throughput emerged in the first decade of blockchain research. But how much concurrency is possible? In their early days, blockchains were mostly used for simple transfers from user to user. More recently, however, *decentralized finance (DeFi)* and *NFT marketplaces* have completely changed what is happening on blockchains. Both are built using *smart contracts* and have gained significant popularity. Transactions on DeFi and NFT marketplaces often interact with the same smart contracts. We believe this development has transformed blockchain usage.
In our work, we perform a historical analysis of Ethereum's transaction graph. We study how much interaction between transactions there was historically and how much there is now. We find that the rise of DeFi and NFT marketplaces has led to an increase in "centralization" in the transaction graph. More transactions are now interconnected: currently there are around 200 transactions per block with 4000 interdependencies between them. We further find that the parallelizability of Ethereum's current interconnected transaction workload is limited. A speedup exceeding a factor of five is currently unrealistic.

Keywords: Blockchain · Ethereum · smart contract · decentralized finance · parallelizability · connectedness · transaction graph.

1 Introduction

When the first blockchain, Bitcoin [33], was launched in 2008, it allowed the execution of financial transactions without relying on a central authority. With its promise, Bitcoin sparked the creation of many more cryptocurrencies, most notably Ethereum [43], which introduced smart contracts in 2015. However, even though cryptocurrencies are continuously reaching new levels of popularity, the transaction throughput of the most popular[1] ones remains incredibly low.

Given the low throughput of blockchains, especially in comparison to established payment systems such as Visa or PayPal, many suggestions to tackle low blockchain throughput levels have been introduced as well as implemented.

---

[1] Measured by total fees users are willing to pay to use the blockchain (see https://cryptofees.info) Ethereum is orders of magnitude more popular than other smart contract-enabled blockchains, such as Avalanche and Cardano.

Layer 2 protocols [13, 36, 25, 34], handling transactions off-chain, sharding protocols [19, 23, 30, 46, 12, 42, 3], and moving from *Proof-of-Work (PoW)* to *Proof-of-Stake (PoS)* [18] are amongst the most adopted scaling solutions. In addition to the development of the aforementioned solutions, the potential of concurrency control for multithreaded execution has been explored thoroughly.

However, these solutions do not focus on the implications of the changing nature of blockchain transactions. Before the rapid rise of *decentralized finance (DeFi)* and NFT marketplaces, transactions were largely simple transactions between two parties. Consequently, dependencies between a large set of transactions in a block were rare. In the face of few dependencies, transaction throughput can be increased with the proposed solutions – as they rely on the parallel execution of transactions to increase throughput. However, DeFi and NFT marketplaces have brought new challenges when scaling throughput.

While DeFi employs smart contracts hosted on the blockchain to offer many of the services provided by traditional finance, NFT marketplaces utilize smart contracts to facilitate NFT purchases. Core smart contracts building DeFi and NFT marketplaces are involved in many of a block's transactions and create dependencies between a significant proportion of transactions. This new reality on Ethereum greatly challenges the parallelization of transaction execution.

In this work, we explore the limits of transaction parallelization on Ethereum. We analyze these limits by investigating the connectedness of the Ethereum mainnet transaction graph over time. The identification of the largest connected component and clique in terms of the required execution workload in a block's transaction graph allows us to explore the potential of concurrent execution over time. In particular, we point out that DeFi's most important smart contracts are central in the transaction graph and responsible for the vast majority of transaction dependencies. Thus, a handful of smart contracts present a significant parallelization bottleneck, especially given the widespread adoption of DeFi and NFT marketplaces starting in 2020.

This development presents a tremendous challenge in the quest to reach throughput levels of established payment systems. We, therefore, conclude by outlining three areas to tackle in order to increase the parallelizability of Ethereum's workload and allow concurrency mechanisms to reach their full potential. These areas should not only be targeted by Ethereum, the focus of our analysis, but also by blockchains with comparable smart contract designs and usage patterns, as we expect them to have similar bottlenecks.

2   Related Work

Blockchain throughput has been one of the first topics of Bitcoin research, and many solutions have emerged to tackle the issue, e.g., layer 2, sharding. Here, we concentrate on those solutions that aim to parallelize the workload through concurrent execution. These works directly study the parallelization of the workload, whose bounds we quantify.

Sergey and Hobor [39] are among the first to explore smart contract concurrency for parallel execution of blockchain transactions. They provide an analogy

between smart contracts and concurrent programming. In the scheme introduced by Zhang and Zhang [48], miners use concurrency control techniques to pre-compute a serializable schedule that can be utilized by validators replaying the block. By employing a dependency graph based concurrency control technique, Amiri et al. [1] find a valid schedule execution that allows for non-conflicting transactions to execute in parallel. Our work, on the other hand, explores the blockchain transaction dependency graph to quantify the existing real-world potential of parallelization.

Additionally, a recent line of work surrounding smart contracts' concurrency leverages speculative execution. Dickerson et al. [15], and Anjana et al. [2] pre-compute a serializable concurrent schedule for a block's transactions through speculative execution, while Gelashvili et al. [21] propose Block-STM, a parallel execution engine that avoids pre-computation. An estimation of the potential concurrency of speculative execution by miners is offered by Saraph and Herlihy [38]. Chen et al. [11] take speculative execution to a new level by speculatively executing transactions that are waiting to be included in a block. In contrast, we explore the limits of concurrency given the nature of blockchain transactions in light of the recent rise of DeFi and NFT marketplaces.

Pîrlea et al. [35] and Murgia et al. [32] utilize static analysis to parallelize execution. They statically determine which transactions can safely be executed in parallel and which contracts can be placed on different shards. While static analysis is valuable to identifying dependencies ahead of time, the existing approaches do not remove inherent dependencies from the workload, which are at the center of our findings.

A parallel line of work studies the transaction graphs of popular cryptocurrencies. Ron and Shamir [37] first analyzed the Bitcoin transaction graph. While their work studies the full transaction graph statically, Kondor et al. [26] also examine changes in the Bitcoin transaction network over time. Several studies also explore the Ethereum transaction network [22, 31, 31, 10, 4, 20, 29, 28, 49, 24, 45, 44, 49] through temporal graph analysis. Instead of solely studying the evolution of Ethereum's full transaction network, we focus on the impacts of the increased smart contract usage on the connectedness of Ethereum's block-wise transaction graphs and quantify the implications for parallelizability of the current and historical transaction workload.

In their study of Ethereum's transaction network, Zanelatto et al. [47] focus on understanding the evolution of connected components in the network. Their work precedes the rise of DeFi and NFT marketplaces on the Ethereum and therefore does not capture the increased trend of more and more interplay between transactions and different smart contracts. Our work focuses on this increased connectivity and further discusses its implications on the blockchain.

3   Background

In the following, we introduce the essential preliminaries concerning transaction execution on the Ethereum blockchain and DeFi smart contracts.

3.1   Ethereum Transaction Execution

Ethereum is a smart contract platform, i.e., it does not only support Ether transfers. Instead, it runs a general purpose virtual machine, the *Ethereum virtual machine (EVM)*, that executes a specific byte code instruction set. Thus, Ethereum allows functions defined in smart contracts to be called in transactions. Smart contract functions can call functions of other smart contracts, generating *internal calls*. The EVM distinguishes between the following different types of function calls: (1) *call* performs operations scoped to the *called* contract's storage, (2) *delegate call* performs operations scoped to the *calling* contract's storage, (3) *static call* is like a regular call but with read-only access to the storage, and (4) *call code* is a deprecated version of delegate call. Thus, all calls are scoped to a specific contract and have either read/write or read-only access to its storage.

We call two transactions *conflicting* or *dependent* if they have calls with the same scope and one of them has write access on that scope. Note that two transactions that interact with the same smart contract might not necessarily touch the same storage cell. However, taking this coarser view on conflicts simplifies large scale analysis and is easier to reason about for smart contract developers.

There are also EVM instructions (BALANCE, EXTCODESIZE, EXTCODE-HASH, EXTCODECOPY), which allow a contract to read global state (including the current Ether balance and smart contract code) of any address, regardless of the current scope. In practice, out of these mostly EXTCODESIZE is used to detect whether an address is a smart contract. In our analysis we disregard potential conflicts, which could be caused by these instructions.

Ethereum introduced the *access list* [7, 8], which specifies a list of addresses and storage keys that the transaction wants to access, giving a gas discount on these accesses. However, it is optional and not yet widely adopted. We found that, out of more than 600 million transactions only 2 million included an access list and their accuracy and completeness is unclear. Building complete and accurate access lists of just addresses would also be easier for smart contract developers and end users. Further, previous work could not gain a significant advantage by parallelizing at the storage key level [38] as opposed to the address level.

Code execution on the EVM is paid for in *gas*, which is automatically converted from the sender's Ether balance. Gas is a measure for how expensive code execution is for validators on Ethereum [17]. Thus, we use gas as a proxy measure for real execution time (including computation and storage accesses). This way assumptions about the underlying implementation, runtime environment, and hardware are kept to a minimum. Further, we define *sequential gas* as the highest cumulative gas cost of any sequential execution in a parallel schedule.

3.2   Decentralized Finance Smart Contracts

DeFi offers many financial services from traditional finance. Instead of relying on intermediaries, DeFi utilizes smart contracts. We elaborate on the functionality of some of DeFi's most important smart contracts in the following.

ERC-20 Tokens are smart contracts that implement fungible tokens, i.e., they can represent anything that can be owned and exchanged in integer quantities, and adhere to a specific interface standard, the ERC-20 (Ethereum Request for Comments 20) [16]. These implement at least a given set of nine functions, including `transfer` (transferring an amount of tokens from the callers address to a given address) and `balanceOf` (getting token balance of a given address).

Routers are implemented by DEXs as a central frontend to their trading interfaces. These routers are stateless, they simply specify via which *liquidity pools*, trading venues for ERC-20 tokens, trades are routed. The router makes the calls to the liquidity pools according to a pre-defined route.

4   Data Collection

We run an Erigon client [27] to collect Ethereum blockchain data. In particular, we query trace data for the whole blockchain history to better understand the parallelizability of the Ethereum mainnet workload, as well as trends over time. Trace data provides us the internal calls executed by each transaction. In the proceeding evaluation, we look at historical data by sampling 65 blocks per day at random over the whole history of Ethereum's mainnet blockchain. With the historical analysis we can observe long term trends in Ethereum's workload and parallelizability thereof. Additionally, we also look at recent data in more detail by considering every single block over the three-month period from 1 June 2022 through 31 August 2022. Through the recent data we hope to get an accurate view of the current state of parallelizability on the Ethereum blockchain.

5   Ethereum Mainnet Workload

In the following we consider changes in the Ethereum mainnet workload over time. In Fig. 1a, we compute and plot the average number of transactions of the blocks in our data set each month along with the 95% confidence interval. Alongside this, we also show the Ether price.

While the number of transactions per block is initially only small, i.e., less than ten transactions, we notice a first significant rise in the number of transactions per block starting in early 2017 and peaking in late 2017 at around 200 transactions per block. During this time, there were unprecedented levels of speculation surrounding cryptocurrencies; this hype likely drove the rapid increase in the Ether price and the number of transactions on the Ethereum mainnet. The market's subsequent cool-down is reflected in a transaction number decrease to around 100 per block and a sharp Ether price decrease. In fact, until the end of 2019, the number of transactions per block and the Ether price are highly correlated, i.e., the Pearson correlation coefficient is 0.77. Later, another increase in the average transaction number occurred in 2020 during the DeFi and the subsequent NFT boom. Since then, the average number of transactions per block was stable at around 200, and the average number of transactions did not significantly surpass the previous peak. With the rise of DeFi and NFTs, the correlation between the number of transactions and the Ether price also decreased, i.e., the Pearson correlation coefficient drops to 0.60 from 2020 onward.
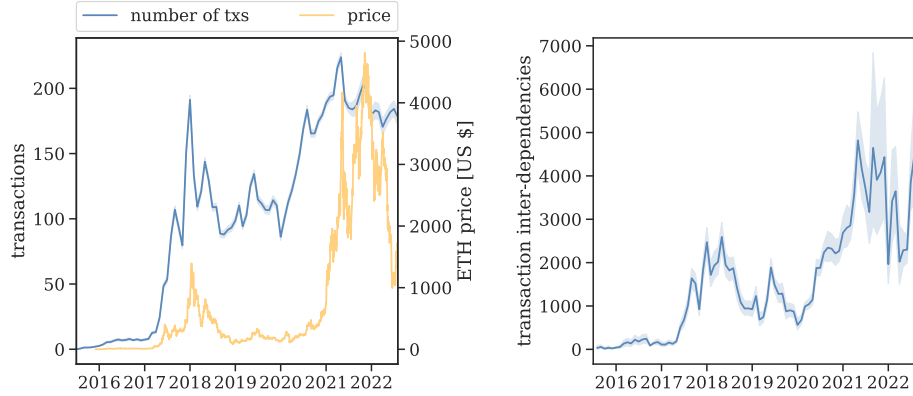
(a) Historical development of the number of transactions per block on Ethereum mainnet and the Ether price.

(b) Historical development of the number of transaction interdependencies per block.

Fig. 1: Visualization of the average number of transactions per block (cf. Fig. 1a) and the average number of transaction interdependencies per block (cf. Fig. 1b). We randomly sample 65 blocks per day and plot the daily average along with the 95% confidence interval.

We further plot the number of transaction interdependencies in Fig. 1b, i.e., the number of transaction pairs that access the same smart contract. While the impact of DeFi and NFT marketplaces on the number of transactions does not significantly surpass previous levels, this does not hold for the number of transaction interdependencies. Notice that there were, on average, around 2000 transaction interdependencies per block during the initial peak. However, the increased usage of smart contracts starting from 2020 also increased the average number of transaction interdependencies per block to 5000 at its peak and never significantly dropped afterward. Note that these interdependencies are largely created by a few core DeFi smart contracts, i.e., popular ERC-20 tokens, DEX routers, and NFT marketplaces. Thus, the widespread adoption of these smart contracts presents a significant challenge to the parallelization of Ethereum transactions.

6   Transaction Graph Representation

To explore trends in parallelization potential we consider a graph representation of the transaction data. There are two graph representations commonly utilized for Ethereum transaction data: *address-based* and *transaction-based*. We provide a definition for the address-based graph in Definition 1 and visualize an example in Fig. 2a. Observe that transaction tx_3 involves four address, $D \to C \to A \to B$, while tx_2 only involves two addresses, $B \to C$.

Definition 1 (Address-based Graph ($\mathbf{AG}_{n,m}$)). *The address-based graph for blocks $n, \ldots, m$, $n \leq m$, is represented as $AG_{n,m}(V, E, \{\omega_e\}_{e \in E})$. Here, V is the graph's set of vertices, each $v \in V$ is an address and V is the set of all*
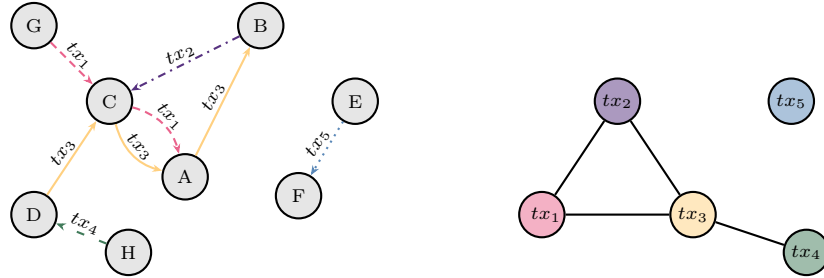
(a) Address-based graph representation of a sample set of Ethereum transactions, where vertices are addresses (contracts or wallets) and edges are calls. The color and style of an edge indicates which transaction the call belongs to.

(b) Transaction-based graph representation of a sample set of Ethereum transactions, where vertices are transactions and the edges indicate dependency between two transactions. Transactions that interact with the same address are dependent.

Fig. 2: Two types of graph representations of the same sample set of five Ethereum transactions. The edge colors indicate belonging to a transaction in the address-based graph representation (cf. Fig. 2a), the transaction-based graph representation has the same transaction set as vertices.

addresses that appeared in blocks $n, \ldots, m$, i.e., were the sender or receiver of one of the (internal) calls of one of the block's transactions. E is the graph's set of edges, each edge $e = (v, u)$ shows a call of contract u by contract v in blocks $n, \ldots, m$. The weight of edge e, ω_e, is given by the amount of gas utilized by the corresponding internal call.

In Definition 2, we define the transaction-based graph and draw the corresponding example in Fig. 2b. Note that the address-based representation induces the transaction-based representation. In the transaction-based graph neighboring transactions cannot safely be executed in parallel. Therefore, in the example shown in Fig. 2b, transaction tx_4 cannot be executed while tx_3 executes as they both interact with address D (cf. Fig. 2a). Cliques in the transaction-based representation indicate that all transactions in the clique have to be executed sequentially. Thus, the execution of transactions tx_1, tx_2 and tx_3 cannot be parallelized. In the following, we will largely consider the address-based representation, but will also draw unique insights from the transaction-based representation, i.e., calculate the graph's biggest clique to explore the limits of parallelization.

Definition 2 (Transaction-based Graph ($\mathbf{TG}_{n,m}$)). *The transaction-based graph for blocks $n, \ldots, m$, $n \leq m$, is represented as $TG_{n,m}(V, E, \{\omega_v\}_{v \in V})$. Here, V is the graph's set of vertices, each $v \in V$ is a transaction and V is the set of all transactions that appeared in blocks $n, \ldots, m$. The weight of a vertex v, ω_v, is the amount of gas utilized by transaction v. E is the graph's set of edges, each edge $e = \{v, u\}$ shows a dependency between transaction v and u in blocks $n, \ldots, m$. A dependency is induced when the two transactions interact with the same address, i.e., the address appears as sender or receiver of an (internal) call for each of the two transactions.*

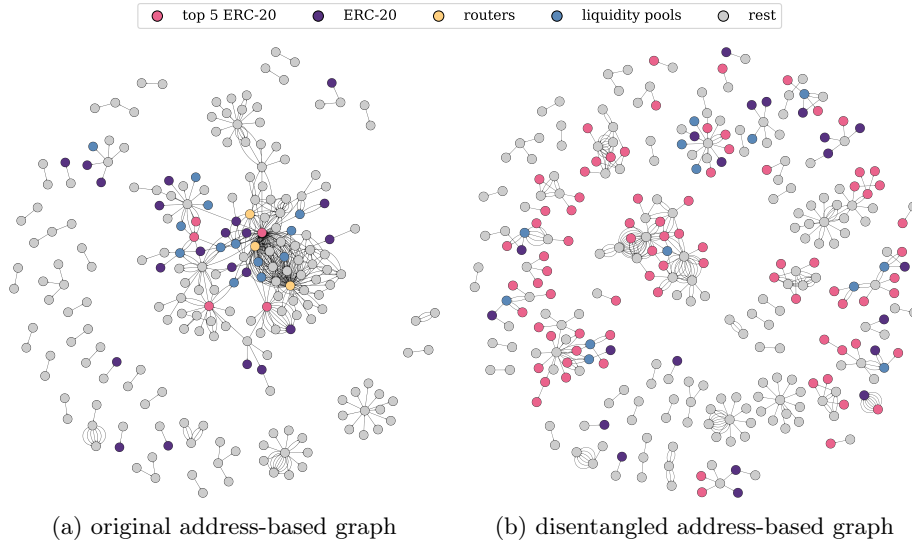(a) original address-based graph   (b) disentangled address-based graph

Fig. 3: Address-based visualisation of block 15,348,042 (mined 15 August 2022). Fig. 3a shows the original graph, while Fig. 3b visualizes a disentangled version of the graph (cf. Section 6.1). In both graphs we highlight some core DeFi smart contracts, namely, ERC-20 tokens, DEX routers, and DEX liquidity pools. Note that the biggest connected component of the disentangled graph is significantly smaller than that of the original graph.

In Fig. 3a we visualize the address interactions of block 15,348,042 (mined 15 August 2022) and highlight some core DeFi smart contracts. The pink vertices are the top five ERC-20 tokens (WETH, USDC, USDT, DAI, and LINK) in terms of the number of transfers, while the purple vertices are the remaining ERC-20 token addresses that appeared in block 15,348,042. We highlight the following DEX routers in yellow: Uniswap V2, Uniswap V3, SushiSwap, and 1inch, and utilize blue to flag the Uniswap V2, Uniswap V3, SushiSwap, and Curve liquidity pools. Notice that the majority of the block builds a single connected component and that the vast majority of the labeled DeFi contracts are part of this connected component. With some, mainly the top 5 ERC-20 and the DEX routers, being central in this connected component and thereby contributing greatly to the dependencies between the different transactions in a block.

6.1 Disentangled Transaction Graph Representation

Observing the persistently high degree (in the address-based transaction graph) of these DeFi contracts across the majority of the blocks since the rise of DeFi, we noticed that many of the dependencies introduced by these smart contracts, which are a central part of the DeFi ecosystem, are by no means essential. Especially the apparent dependencies introduced by ERC-20 token contracts and DEX routers in the smart contract level, would not manifest in the storage key level. These two examples of non-essential dependencies are relatively easy to spot by validators as we outline in the following.
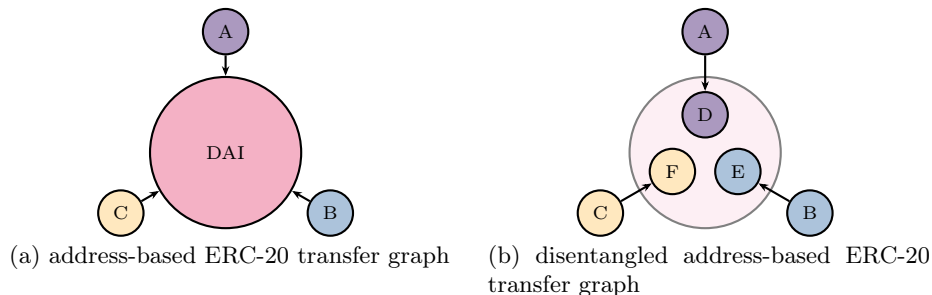
(a) address-based ERC-20 transfer graph

(b) disentangled address-based ERC-20 transfer graph

Fig. 4: Address-based graph representation of three transfers of DAI, an ERC-20 token, between the following wallets: address A to address D, address B to address E, and address C to address F. In Fig. 4a we show the actual address-based representation. Observe that the three transfers appear dependent of each other. In Fig. 4b we show how we disentangle the graph to avoid this dependency.

For example, consider three transfers of DAI, an ERC-20 token, between the following addresses: address A to address D, address B to address E, and address C to address F. As DAI is not the Ethereum blockchain's native currency, we only observe calls from the DAI senders to the DAI smart contract, which keeps track of fungible DAI tokens (cf. Fig. 4a). Thus, the three transfers appear dependent, which would not be the case for three equivalent ETH transfers. Therefore, we disentangled the transaction graph representation in Fig. 4b. Instead of having the transaction's sender call the token contract, we pretend that they call the memory location of the receiver in the DAI smart contract. In addition to making these adjustments for the ERC-20 `transfer` function, we also make respective adjustments for the following ERC-20 contract functions: `balanceOf`, `transferFrom`, `approve`, and `allowance`. Note that we only perform this disentanglement for the top five ERC-20 tokens: WETH, USDC, USDT, DAI, and LINK. These five ERC-20 tokens are the five largest in terms of the number of transfers and together account for 34% of all ERC-20 transfers. Further, we choose to restrict ourselves to this small number of ERC-20 tokens to show: (1) their impact on the connectedness of a block's transactions and (2) ensure that the tokens do not have any unexpected behavior, e.g., transferring a proportion to a third party [14].

DEX routers are also involved in many transactions and, thereby, lead to increased connectedness in the transaction graph. As the DEX routers themselves are stateless and only perform calls to the indicated liquidity pools on the user's behalf, the dependencies in the smart contract level are therefore not necessary. Thus, we remove routers from the transaction graph. In particular, we re-route all the router edges to the sender of the respective transaction. In the later analysis, we perform this disentanglement for the routers of the following DEXs: Uniswap V2, Uniswap V3, SushiSwap, and 1inch.

7   Parallelizability

In the following exploration of Ethereum's transaction graphs, we quantify the limited parallelization potential. In Section 7.1, we discuss the evolution over
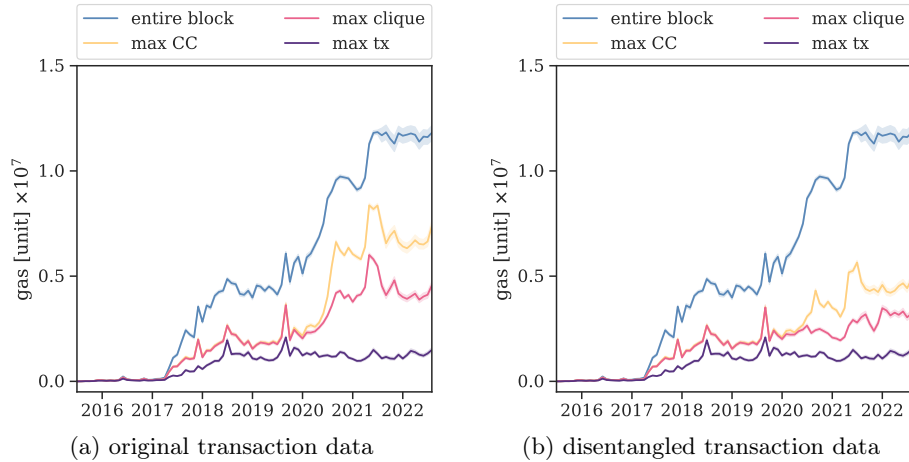
Fig. 5: We plot the gas used by: (1) the entire block, (2) the block's heaviest connected component (CC), (3) the block's heaviest clique, and (4) the block's heaviest transaction. Fig. 5a analyzes the original transaction data and Fig. 5b the disentangled transaction data. Note that we plot the monthly average along with the 95% confidence interval from randomly sampling 65 blocks per day.

time – cementing the impact of DeFi and NFTs – and, in Section 7.2, the current state of parallelization potential on the Ethereum mainnet. We define *parallelizability* of a block as the highest speedup factor (total gas used by the block divided by sequential gas of a schedule) that can be achieved. For our analysis we look at specific schedules as well as graph metrics, which serve as upper and lower bounds on the parallelizability under our definition of conflicts.

7.1   Parallelizability over Time

In the following, we analyze the parallelization potential on Ethereum's mainnet by considering the connectedness of the transaction graphs. We randomly sample 65 blocks per day over the entire blockchain history up until the last block on 31 August 2022 – allowing us to observe the trends over time.

The adoption of DeFi and NFT marketplaces is clearly visible when looking at trends over time in gas usage, a proxy for the execution time. In Fig. 5a, we plot the amount of gas per block in blue. Notice the sharp increase starting in 2020 with the rise of DeFi. Whereas these new applications did not increase the number of transactions to unprecedented levels, they caused the amount of gas per block to skyrocket due to the increasing complexity of transactions.

To provide an enhanced understanding of the parallelizability of these increasingly heavy blocks, we also plot the size of the heaviest connected component, drawn in yellow, and the size of the heaviest clique, shown in pink, in Fig. 5a. Note that we measure the weight of a connected component or clique by the total amount of gas used by its transactions. Thereby, these weights indicate the time required to execute the contained transactions.

Connected components are equivalent in the address-based and transaction-based graph representations. Across both views, the weight of the heaviest connected component offers a lower bound for the parallelization potential of a block's execution. Any schedule that runs as many transactions as possible in parallel, i.e., in each time step executes a maximal independent set of transactions, will not exceed the time required to execute the heaviest connected component sequentially. We want to note that to obtain the dependencies between transactions, one has to have access to a statically provided access list or first execute all transactions. However, this is done only once by the validator. Once the block was executed, a parallel schedule could be made available to everyone else for validation. Further, we utilize the transaction-based graph representation to find the heaviest clique. For this there is no direct analogue in the address-based graph. The heaviest clique specifies an upper bound for the parallelization potential of a block's execution. Any schedule must handle all transactions in a clique sequentially – assuming atomic transaction execution. We, in fact, ran a simple list scheduling algorithm to find a schedule. It generates a partial ordering and always allows execution of a maximal independent set in parallel. We find that, while the schedule occasionally requires longer to execute than the heaviest clique would, the relative error is negligible (cf. Appendix A). Thus, the upper bound of the parallelization potential is almost achievable with a simple schedule. Note that our transaction graph might over estimate dependencies as we are coming from the smart contract level and not the storage key level.

Looking at our data, we observe that the heaviest connected component currently makes up more than half of the block (cf. Fig. 5a). Further, the difference between the average heaviest connected component and the average heaviest clique has grown since the popularization of DeFi in 2020. This could be explained by interactions between the different protocols and smart contracts of the DeFi ecosystems. Since the rise of DeFi, the transactions in the heaviest connected component tend to interact with popular ERC-20 tokens, DEX liquidity pools, and lending protocols. However, mostly those that interact with the same smart contract are in a clique. We note that the largest clique typically consists of those transactions that interact with WETH, i.e., the most popular ERC-20 token (in terms of the number of transfers).

As previously stated, transactions interacting with WETH generally form the heaviest clique in the original transaction data. However, as we show in Section 6.1, the apparent dependencies in the smart contract level view are simply a consequence of implementing ERC-20 tokens as smart contracts as opposed to native tokens. To obtain a more accurate picture, we perform the previously outlined disentanglement, we observe a significant reduction in both the size of the heaviest connected component and clique (cf. Fig. 5b) since the adoption of DeFi. In fact, from 1 July 2020 to 31 August 2022, the disentanglement decreased the size of the heaviest connected component by a factor of 1.78 on average. Further, the size of the heaviest clique decreased by a factor of 1.88 on average.

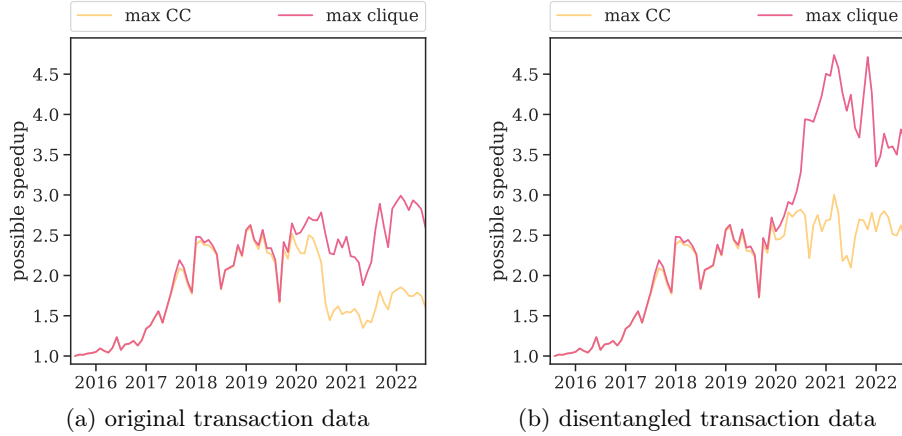(a) original transaction data   (b) disentangled transaction data

Fig. 6: We visualize the achievable execution speedup (aggregated monthly) through parallelization for the original transaction data (cf. Fig. 6a) and the disentangled transaction data (cf. Fig. 6b). We obtain the lower bound for the parallelization potential through the identification of the heaviest connected component and the upper bound from the heaviest clique. Note both the heaviest connected component and clique are weighted by gas.

We also plot the size, in terms of gas used, of the heaviest transaction per block in both Fig. 5a and Fig. 5b.[2] The size of the heaviest transaction in a block indicates a further, looser upper bound for the parallelization potential that disregards any dependencies between transactions. By neglecting all dependencies, we automatically omit any nonessential dependencies. This looser upper bound thus only assumes that transactions must execute atomically. However, we find that, on average, the heaviest transactions are a significant proportion of the entire block – a ninth on average over the entire history. Thus, parallelization is not only limited by the ever-increasing size of the heaviest clique but is similarly bounded by individual large transactions.

We plot the lower and (realistic) upper bound for the achievable speedup in Fig. 6. In Fig. 6a, we show these bounds for the original transaction data, and in Fig. 6b, we show the same bounds for the disentangled transaction data. Similar to our previous observations, the lower (given by the size of the heaviest connected component) and the upper bound (given by the size of the heaviest clique) of the realistically achievable speedup are close to each other up until the rise of DeFi in 2020. Further, we observe the performed disentanglement also only shows its effects from 2020 onward, as it targets DeFi smart contracts. From 2020, we notice an increase in the difference between the lower and upper bound of the achievable speedup. Further, in the original transaction data, we observe that both the lower and the upper bound for the realistically achievable speedup decrease once DeFi becomes adopted (cf. Fig. 6a). In the disentangled

---

[2] Note that the disentanglement does not impact the size of the heaviest transaction, and neither the total gas of a block.

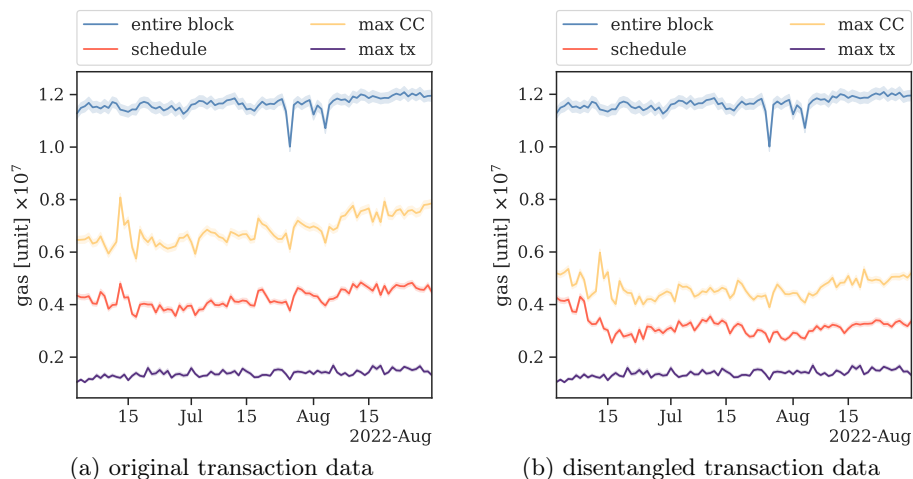(a) original transaction data   (b) disentangled transaction data

Fig. 7: We plot the gas used by: (1) the entire block, (2) the block's heaviest connected component (CC), (3) our naive schedule (sequentially), and (4) the block's heaviest transaction. Fig. 7a analyzes the original transaction data and Fig. 7b the disentangled transaction data. Note that we plot the daily average along with the 95% confidence interval. Further, we use the sequential gas of our schedule as a proxy for the size of the heaviest clique.

transaction data, on the other hand, we notice that the lower bound for the possible speedup does not decrease after the introduction of DeFi, but instead remains more or less constant (cf. Fig. 6b). It is even more remarkable that the upper bound for the realistically achievable speedup even increases after the introduction of DeFi for the disentangled transaction data. We presume this stems from the increasing number of transactions in the same period (cf. Fig. 1a). Further, it is likely impacted by most DeFi transactions being dependent on each other over a given number of hops in the graph representation but not necessarily being all in one clique.

7.2 Current Limits of Parallelizability

To better gauge the current limits of parallelizability, we expand on the previous analysis by analyzing all blocks from 1 June 2022 to 31 August 2022 – allowing us to obtain a complete picture of the current state of the Ethereum mainnet. In Fig. 7, we plot the amount of gas used by: (1) the entire block, (2) the block's heaviest connected component, (3) our schedule (sequentially), and (4) the block's heaviest transactions. Note that we only plot the sequential gas used by our schedule and not the heaviest clique, as finding the heaviest clique is time intensive. Our analysis in Appendix A shows that our schedule almost reaches the same parallelization potential.

When examining Fig. 7, we notice that there are few fluctuations in both the daily mean size of the blocks and the daily mean size of the heaviest transactions. We only observe two collapses, of around 10%, in the mean size of the entire block at the end of July and the beginning of August. When looking at the daily

average size of the heaviest connected component and the daily average amount of sequential gas used by our schedule in the original transaction data (cf Fig. 7a), a similar picture paints itself. In general, both averages make up approximately a half (connected component) and a third (schedule) of the block size on average. There is one peak in the average gas used by the heaviest connected component and the schedule around 15 June 2022 that we do not observe in the block size. The daily price movements of Ether were very high during this time (cf. Appendix C, Fig. 12b) due to the anticipation of and the release of the CPI data [41]. As a consequence, the Ether trading volume on DEXs like Uniswap V3 experienced a rapid increase [40], which we presume lead to an increased size of both the heaviest connected component and clique in relation to the block size. We want to point out that, while the 95% confidence interval is tight around the daily mean for all four graphs, the fluctuations of values for all four graph measures are substantial as shown in Appendix B (cf. Fig. 11). For instance, shortly around the time at which we observe the peak in gas usage, the 99th percentile of the heaviest transaction reaches is almost the 99th percentile of the block size. Thus, there are some blocks in which a single transaction makes up almost the entire block – allowing for little to no parallelization in those blocks. Still, we observe that the daily average of gas usage by the heaviest connected component and by the heaviest clique, for which we use our schedule as a proxy, make up a relatively stable proportion of the block in the recent (original) transaction data.

Turning to the disentangled transaction data (cf. Fig. 7b), we notice a stable reduction in the daily average of the gas used by the heaviest connected components (by a factor of 1.6) and the sequential gas used by the schedule (by a factor of 1.5). It is most remarkable that the reduction is less significant in early June than in the remaining data set. We presume that this is a consequence of the significant price drop of Ether in the same period (cf. Fig. 12a), which likely led to exceptional DeFi usage patterns that further interconnected the workload. In the remaining data set, the reduction achieved by the disentanglement is very stable, but the achievable speedup still only reaches around a factor four (cf. Appendix B, Fig. 10).

Finally, in order to simulate higher transaction throughput, we consider batches of ten consecutive blocks and explore the connectedness of the corresponding transaction graphs (cf. Fig. 8). Even with this (rather exaggerated) simulated increase in block size, the sequential gas of our schedule increases proportionally, thus not changing this upper bound for the realistically achievable speedup. On the other hand, the lower bound (indicated by the heaviest connected component) even becomes much looser. This suggests that, when merging blocks, the largest cliques of all blocks merge into one, whereas connected components are even merged from within the same block. This is in line with our analysis that the heaviest cliques are always induced by the same few contracts – indicating that increasing block size does not improve concurrency potential.

8   Discussion and Conclusion

Our work quantifies the parallelizability of the Ethereum mainnet workload.

(a) original transaction data   (b) disentangled transaction data
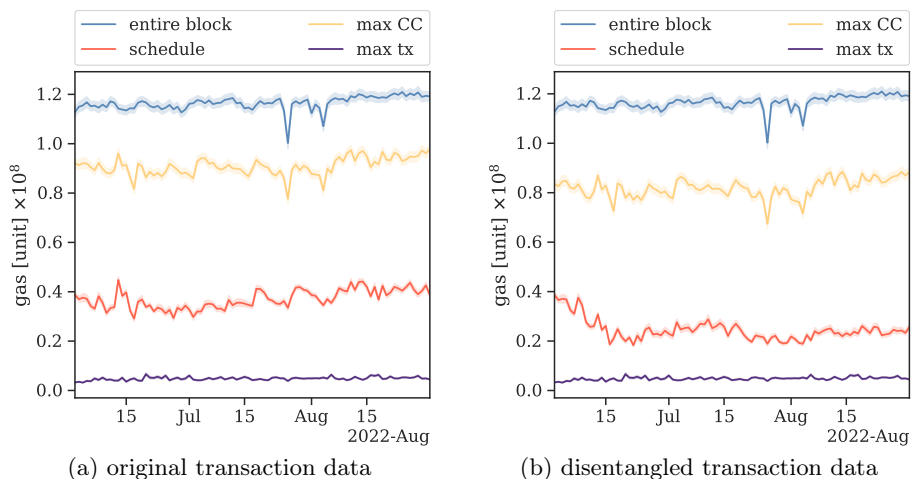
Fig. 8: We plot the gas used by: (1) 10 consecutive blocks, (2) their heaviest connected component (CC), (3) our naive schedule (sequentially), and (4) their heaviest transaction. Fig. 8a analyzes the original transaction data and Fig. 8b the disentangled transaction data. Note that we plot the daily average along with the 95% confidence interval.

We find that currently, the level of concurrency is very limited. Thus, it does not suffice to only devote efforts to finding ways to best exploit the existing potential for concurrency. Instead, we believe that part of the focus must be shifted towards ensuring that the workload is parallelizable in the first place. Concretely, we believe that the following three areas must be targeted to enable the existing concurrency mechanisms to achieve much higher speedups.

Investigate dependencies. As we outline, some of DeFi's core smart contracts appear in many transactions. Thus, we believe that transaction dependencies must be investigated on a more fine-grained basis, for example at the storage key level. Furthermore, the smart contracts could be redesigned to avoid unnecessary dependencies in the transaction graph.

Incentivize "simple" transactions. The heaviest transaction in a block currently makes up around one tenth of the average block size. Thus, these individual transactions present a limit on the parallelization potential. We therefore believe that the blockchain should discourage such frequent heavy transactions and instead encourage simple transactions. One possible approach would be charging for computation superlinearly.

Increase predictability of dependencies. The incredibly low usage of the access list, indicates that it is currently not viable for transaction senders to provide the addresses and storage keys their transaction will touch. Predictability of dependencies would take care of this situation and would allow for increased parallelization during execution.

Only once the workload on the Ethereum mainnet is truly parallelizable will the speedup suffice to make the 100,000 transactions per second [9] stated by the Ethereum Foundation achievable.

References

1. Amiri, M.J., Agrawal, D., El Abbadi, A.: Parblockchain: Leveraging transaction parallelism in permissioned blockchain systems. In: 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS). pp. 1337–1347. IEEE (2019)
2. Anjana, P.S., Kumari, S., Peri, S., Rathor, S., Somani, A.: Optsmart: a space efficient optimistic concurrent execution of smart contracts. Distributed and Parallel Databases pp. 1–53 (2022)
3. Avarikioti, G., Kokoris-Kogias, E., Wattenhofer, R.: Divide and scale: Formalization of distributed ledger sharding protocols. arXiv preprint arXiv:1910.10434 (2019)
4. Bai, Q., Zhang, C., Xu, Y., Chen, X., Wang, X.: Evolution of ethereum: A temporal graph perspective. arXiv preprint arXiv:2001.05251 (2020)
5. Berg, J.A., Fritsch, R., Heimbach, L., Wattenhofer, R.: An Empirical Study of Market Inefficiencies in Uniswap and SushiSwap. In: The 2nd Workshop on Decentralized Finance (DeFi), Grenada (February 2022)
6. Buterin, V., Conner, E., Dudley, R., Slipper, M., Norden, I., Bakhta, A.: EIP-1559. https://eips.ethereum.org/EIPS/eip-1559 (2022)
7. Buterin, V., Swende, M.: Eip-2929: Gas cost increases for state access opcodes. https://eips.ethereum.org/EIPS/eip-2929 (2022)
8. Buterin, V., Swende, M.: Eip-2930: Optional access lists. https://eips.ethereum.org/EIPS/eip-2930 (2022)
9. Cavicchioli, M.: Ethereum will reach 100,000 transactions per second. https://en.cryptonomist.ch/2022/07/22/ethereum-reach-100000-transactions-second/ (2022)
10. Chen, T., Li, Z., Zhang, Y., Luo, X., Chen, A., Yang, K., Hu, B., Zhu, T., Deng, S., Hu, T., et al.: Dataether: Data exploration framework for ethereum. In: 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS). pp. 1369–1380. IEEE (2019)
11. Chen, Y., Guo, Z., Li, R., Chen, S., Zhou, L., Zhou, Y., Zhang, X.: Forerunner: Constraint-based speculative transaction execution for ethereum. In: Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles. pp. 570–587 (2021)
12. 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)
13. Decker, C., Wattenhofer, R.: A Fast and Scalable Payment Network with Bitcoin Duplex Micropayment Channels. In: 17th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS), Edmonton, Canada (August 2015)
14. DeFi Cartel: Salmonella. https://github.com/Defi-Cartel/salmonella (2022)
15. Dickerson, T., Gazzillo, P., Herlihy, M., Koskinen, E.: Adding concurrency to smart contracts. In: Proceedings of the ACM Symposium on Principles of Distributed Computing. pp. 303–312 (2017)
16. Ethereum Foundation: ERC-20 token standard. https://ethereum.org/en/developers/docs/standards/tokens/erc-20/ (2022)
17. Ethereum Foundation: Gas and fees. https://ethereum.org/en/developers/docs/gas (2022)
18. Ethereum Foundation: The merge. https://ethereum.org/en/upgrades/merge/ (2022)

19. Ethereum Foundation: Sharding. https://ethereum.org/en/upgrades/sharding/ (2022)
20. Ferretti, S., D'Angelo, G.: On the ethereum blockchain structure: A complex networks theory perspective. Concurrency and Computation: Practice and Experience **32**(12), e5493 (2020)
21. Gelashvili, R., Spiegelman, A., Xiang, Z., Danezis, G., Li, Z., Xia, Y., Zhou, R., Malkhi, D.: Block-stm: Scaling blockchain execution by turning ordering curse to a performance blessing. arXiv preprint arXiv:2203.06871 (2022)
22. Guo, D., Dong, J., Wang, K.: Graph structure and statistical properties of ethereum transaction relationships. Information Sciences **492**, 58–71 (2019)
23. Han, R., Yu, J., Zhang, R.: Analysing and improving shard allocation protocols for sharded blockchains. Cryptology ePrint Archive (2020)
24. He, N., Su, W., Yu, Z., Liu, X., Zhao, F., Wang, H., Luo, X., Tyson, G., Wu, L., Guo, Y.: Understanding the evolution of blockchain ecosystems: A longitudinal measurement study of bitcoin, ethereum, and eosio. arXiv preprint arXiv:2110.07534 (2021)
25. Kalodner, H., Goldfeder, S., Chen, X., Weinberg, S.M., Felten, E.W.: Arbitrum: Scalable, private smart contracts. In: 27th USENIX Security Symposium (USENIX Security 18). pp. 1353–1370 (2018)
26. Kondor, D., Pósfai, M., Csabai, I., Vattay, G.: Do the rich get richer? an empirical analysis of the bitcoin transaction network. PloS one **9**(2), e86197 (2014)
27. ledgerwatch: Erigon. https://github.com/ledgerwatch/erigon (2022)
28. Lin, D., Chen, J., Wu, J., Zheng, Z.: Evolution of ethereum transaction relationships: Toward understanding global driving factors from microscopic patterns. IEEE Transactions on Computational Social Systems **9**(2), 559–570 (2021)
29. Lin, D., Wu, J., Yuan, Q., Zheng, Z.: Modeling and understanding ethereum transaction records via a complex network approach. IEEE Transactions on Circuits and Systems II: Express Briefs **67**(11), 2737–2741 (2020)
30. Luu, L., Narayanan, V., Zheng, C., Baweja, K., Gilbert, S., Saxena, P.: A secure sharding protocol for open blockchains. In: Proceedings of the 2016 ACM SIGSAC conference on computer and communications security. pp. 17–30 (2016)
31. Motamed, A.P., Bahrak, B.: Quantitative analysis of cryptocurrencies transaction graph. Applied Network Science **4**(1), 1–21 (2019)
32. Murgia, M., Galletta, L., Bartoletti, M.: A theory of transaction parallelism in blockchains. Logical Methods in Computer Science **17** (2021)
33. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Decentralized Business Review p. 21260 (2008)
34. Optimism Foundation: Optimism. https://www.optimism.io (2022)
35. Pîrlea, G., Kumar, A., Sergey, I.: Practical smart contract sharding with ownership and commutativity analysis. In: Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation. pp. 1327–1341 (2021)
36. Poon, J., Dryja, T.: The bitcoin lightning network: Scalable off-chain instant payments (2016)
37. Ron, D., Shamir, A.: Quantitative analysis of the full bitcoin transaction graph. In: International Conference on Financial Cryptography and Data Security. pp. 6–24. Springer (2013)
38. Saraph, V., Herlihy, M.: An empirical study of speculative concurrency in ethereum smart contracts. arXiv preprint arXiv:1901.01376 (2019)

39. Sergey, I., Hobor, A.: A concurrent perspective on smart contracts. In: International Conference on Financial Cryptography and Data Security. pp. 478–493. Springer (2017)
40. Uniswap Labs: Ether. https://info.uniswap.org/#/tokens/0xc02aaa39b223fe8d0a0 e5c4f27ead9083c756cc2 (2022)
41. U.S. Bureau of Labor Statistics: Schedule of releases for the consumer price index. https://www.bls.gov/schedule/news_release/cpi.htm (2022)
42. Wang, G., Shi, Z.J., Nixon, M., Han, S.: SoK: Sharding on blockchain. In: Proceedings of the 1st ACM Conference on Advances in Financial Technologies. pp. 41–61 (2019)
43. Wood, G., et al.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper **151**(2014), 1–32 (2014)
44. Xie, Y., Jin, J., Zhang, J., Yu, S., Xuan, Q.: Temporal-amount snapshot multigraph for ethereum transaction tracking. In: International Conference on Blockchain and Trustworthy Systems. pp. 133–146. Springer (2021)
45. Xie, Y., Zhou, J., Wang, J., Zhang, J., Sheng, Y., Wu, J., Xuan, Q.: Understanding ethereum transactions via network approach. In: Graph data mining, pp. 155–176. Springer (2021)
46. 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)
47. Zanelatto Gavião Mascarenhas, J., Ziviani, A., Wehmuth, K., Vieira, A.B.: On the transaction dynamics of the ethereum-based cryptocurrency. Journal of Complex Networks **8**(4), cnaa042 (2020)
48. Zhang, A., Zhang, K.: Enabling concurrency on smart contracts using multiversion ordering. In: Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint International Conference on Web and Big Data. pp. 425–439. Springer (2018)
49. Zhao, L., Sen Gupta, S., Khan, A., Luo, R.: Temporal analysis of the entire ethereum blockchain network. In: Proceedings of the Web Conference 2021. pp. 2258–2269 (2021)

A   Relative Error of Schedule

We compare the sequential gas used by our naive schedule to the amount of gas utilized for the execution of the heaviest clique in Fig. 9. Notice that the mean daily relative error is very small, less than 1%, throughout the entire blockchain history. The relative error peaks for both the original transaction data and the disentangled transaction data, starting with the rise of DeFi up until mid-2021. From mid-2021 onward, it becomes even smaller again. We further point out that the relative error of our schedule is slightly larger for the disentangled transaction data than for the original data. Regardless, the sequential gas utilized by our naive schedule accurately approximates the amount of gas utilized for the execution of the heaviest clique, which is optimal for any schedule working with conflicts on the granularity of addresses.

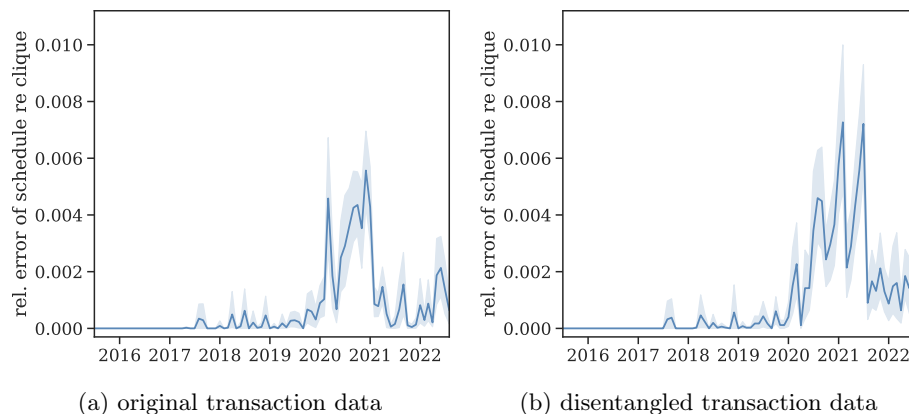(a) original transaction data   (b) disentangled transaction data

Fig. 9: We visualize the relative error between the sequential gas of the schedule obtained through our naive scheduling algorithm and the heaviest clique, which is the best achievable for any parallel schedule working on the address level. We plot daily average relative error of 65 randomly sampled blocks per day and 95% confidence interval for the original transaction data (cf. Fig. 9a) and disentangled transaction data (cf. Fig. 9b).

B   Current Limits of Parallelizability

We plot bounds for the achievable execution speedup of the gas utilized for the execution of the entire block (averaged over each day), in Fig. 10a for the original transaction data and in Fig. 10b for the disentangled transaction data. The heaviest connected component, in terms of gas used, serves as a lower bound, whereas the sequential gas from our naive scheduling algorithm, which as we saw closely approximates an optimal schedule, serves as a proxy for the upper

(a) original transaction data
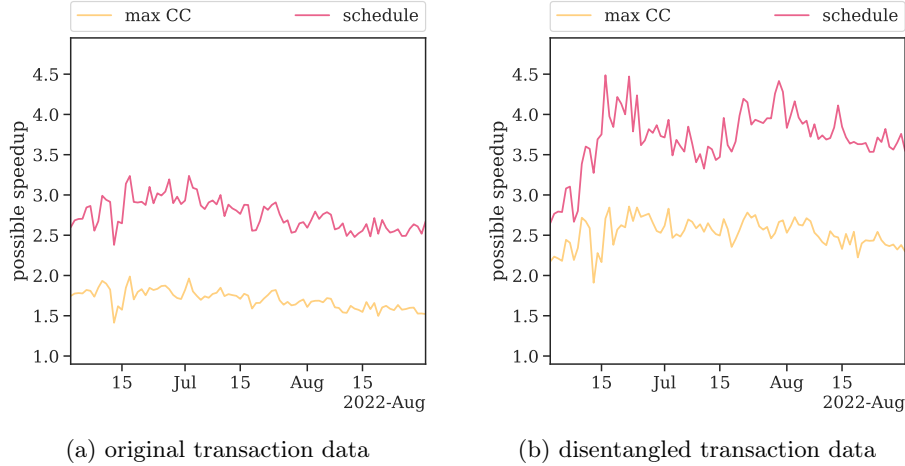
(b) disentangled transaction data

Fig. 10: We visualize the achievable execution speedup (aggregated daily) through parallelization for the original transaction data (cf. Fig. 10a) and the disentangled transaction data (cf. Fig. 10b). We obtain the lower bound for the parallelization potential through the identification of the heaviest connected component and the upper bound from the heaviest clique. Note that both the heaviest connected component and clique are weighted by gas.

bound. We see that a speedup of 4.5x is not surpassed even including our disentanglement, even though we only look at theoretical bounds and do not impose limits on the degree of parallelism, e.g. a maximum number of threads.

We plot the 90th percentile of the gas utilized for (1) the sequential execution of the entire block, (2) the heaviest connected component, (3) our naive schedule, and (4) the heaviest transaction, in Fig. 11a for the original transaction data and in Fig. 11b for the disentangled transaction data. Notice that in comparison to the daily mean (cf. Fig. 7a), the daily 90th percentile of gas used by all four measures is approximately a factor of two larger. Thus, 10% of blocks are twice the size of the average block. Note that EIP-1559 permits blocks to be at most twice the size of the target block size [6], and we are observing a significant proportion of blocks being built with a size very close to twice the target size. We also observe many blocks that are almost empty at an equal rate and presume that this pattern of empty and overfull blocks emerged to manipulate the base fee introduced in EIP-1559. The fact that we observe a similar increase in the 90th percentile of gas used by the heaviest connected component and our naive schedule, in comparison to the mean, indicates that in the 90th percentile of blocks, in terms of gas used, the level of connectedness is similar to that in an average block. That is, the proportion of a block in the heaviest connected component and clique remains largely unchanged. Note that this is the case for both the original and disentangled transaction data.

If we turn to the 99th percentile of gas usage (cf. Figs. 11c and 11d), a different picture paints itself. We are no longer simply observing a proportional
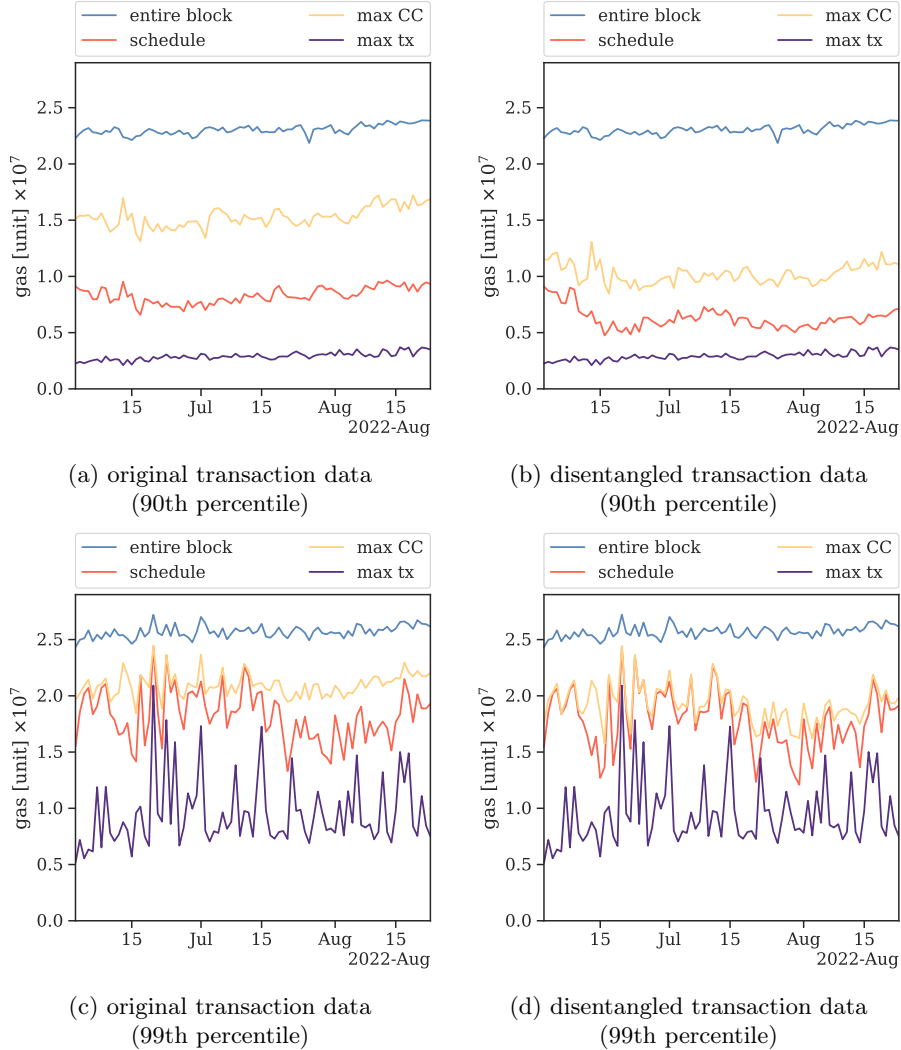
(a) original transaction data
(90th percentile)

(b) disentangled transaction data
(90th percentile)

(c) original transaction data
(99th percentile)

(d) disentangled transaction data
(99th percentile)

Fig. 11: We plot daily 90th and 99th percentile of the gas used by: (1) the entire block, (2) the block's heaviest connected component (CC), (3) our naive parallel schedule (sequential gas), and (4) the block's heaviest transaction. Figs. 11a and 11c analyze the original transaction data and Figs. 11b and 11d analyze the disentangled transaction data. Note that we use the sequential gas utilized by our schedule as a proxy for the size of the heaviest clique.

increase in gas usage across all four measures but instead observe widely different patterns. Only the 99th percentile gas usage of the entire block appears stable, as it is limited by EIP-1559. The 99th percentile of both the heaviest connected component and the heaviest clique is significantly larger than the average block size (cf. Fig. 7). This holds for both the original and the disentangled transaction

data. In the 99th percentile, our disentanglement has little impact on the heaviest cliques but still reduces the size of the largest connected components slightly. We presume that this is due to the different nature of the transactions in the heaviest clique in these extraordinary cases. Further, the 99th percentile of the largest transaction is as large as the mean block size (cf. Fig. 7) and half the size of the 99th percentile of the block size. Thus, the largest transactions make up significant proportions of a block and thereby hugely limit the parallelization potential of those blocks.

C   Recent Ether Price Data

In Fig. 12, we plot Ether's daily price and daily price movement from 1 June 2022 to 31 August 2022. Fig. 12a visualizes the Ether's daily open price (p_{open}) and daily close price (p_{close}). The daily price movement, which we plot in Fig. 12b, is a measure of the daily price volatility and is given by

$$\frac{p_{high} - p_{low}}{p_{low}},$$

where p_{high} is the day's highest Ether price and p_{low} is the day's lowest Ether price [5].

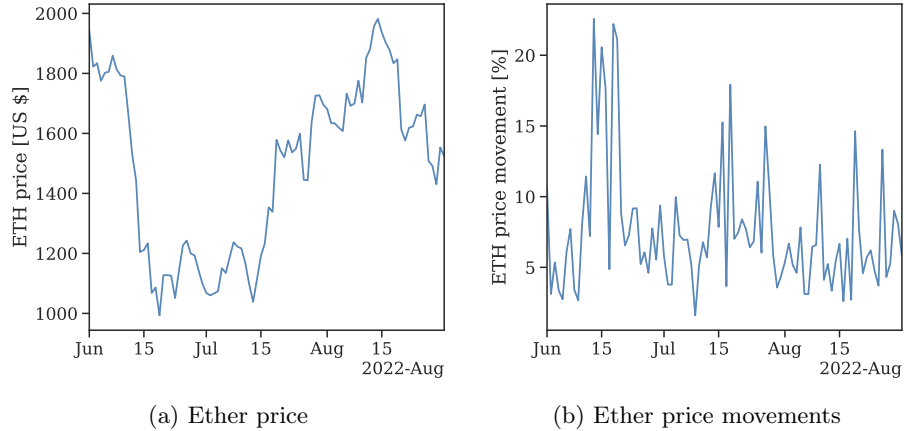(a) Ether price   (b) Ether price movements

Fig. 12: We plot the Ether price and daily price movement from 1 June 2022 through 31 August 2022. The daily price movement compares the day's high to the day's low and is, therefore, a measure of price volatility.