Transaction Fee Market Design for Parallel Execution

BAHAR ACILAN, ETH Zürich, Switzerland
ANDREI CONSTANTINESCU, ETH Zürich, Switzerland
LIOBA HEIMBACH, ETH Zürich, Switzerland
ROGER WATTENHOFER, ETH Zürich, Switzerland

Given the low throughput of blockchains like Bitcoin and Ethereum, scalability — the ability to process an increasing number of transactions — has become a central focus of blockchain research. One promising approach is the parallelization of transaction execution across multiple threads. However, achieving efficient parallelization requires a redesign of the incentive structure within the fee market. Currently, the fee market does not differentiate between transactions that access multiple high-demand resources versus a single low-demand one, as long as they require the same computational effort. Addressing this discrepancy is crucial for enabling more effective parallel execution.

In this work, we aim to bridge the gap between the current fee market and the need for parallel execution by exploring alternative fee market designs. To this end, we propose a framework consisting of two key components: a *Gas Computation Mechanism (GCM)*, which quantifies the load a transaction places on the network in terms of parallelization and computation, measured in *units of gas*, and a *Transaction Fee Mechanism (TFM)*, which assigns a price to each unit of gas. We also introduce a set of desirable properties for a GCM, present multiple candidate mechanisms, and evaluate them against the properties. One promising candidate emerges: the *weighted area GCM*. Notably, this mechanism can be seamlessly composed with existing TFMs, such as EIP-1559. While our exploration primarily focuses on the execution component of the fee, which directly relates to parallel execution, we also outline how it could be integrated with fees associated with other factors, such as storage and data bandwidth, by drawing a parallel to a multi-dimensional fee market.

Authors' Contact Information: Bahar Acilan, bacilan@ethz.ch, ETH Zürich, Switzerland; Andrei Constantinescu, aconstantine@ethz.ch, ETH Zürich, Switzerland; Lioba Heimbach, hlioba@ethz.ch, ETH Zürich, Switzerland; Roger Wattenhofer, wattenhofer@ethz.ch, ETH Zürich, Switzerland.

1 Introduction

Scalability — the ability to process more transactions efficiently — has become a central focus in blockchain research, especially given the low throughput of many existing networks. Ethereum, for example, is constrained by its single-threaded execution model, limiting transaction throughput. One promising way to enhance scalability is by parallelizing transaction execution across multiple threads, taking advantage of the multi-core processors common in modern hardware. However, achieving the full efficiency gains of parallel execution requires rethinking fee market design to better account for resource contention and scheduling constraints.

A *transaction fee mechanism (TFM)* is a core component of any blockchain protocol, determining which pending transactions should be processed and what users must pay for the privilege of having their transactions executed. Traditional fee mechanisms, like Bitcoin's first-price auction, involve users submitting bids with their transactions, and the transactions with the highest bids per computation are included in the next block. Ethereum initially used a similar purely first-price auction-based model but switched to the more sophisticated EIP-1559 mechanism in 2021, which introduces a fluctuating base fee based on network demand, aiming to improve incentive compatibility and reduce price volatility [10]. Most of the existing literature on transaction fee mechanisms focuses on settings where transactions are executed sequentially and therefore does not account for resource contention, which is crucial in parallel execution environments.

Thus, viewed in isolation, these traditional TFMs are ill-suited to the complexities introduced by parallel transaction execution. They price transactions solely based on their computational cost, without distinguishing between those that access a single resource and those that interact with multiple, potentially contested resources. This pricing model works in a single-threaded environment but fails to capture the nuances of parallel execution, where transactions may impose vastly different constraints on resource scheduling. Transactions that touch multiple high-contention resources can introduce bottlenecks, while those interacting with isolated resources are far easier to schedule efficiently.

Ethereum has yet to adopt parallel execution, but several blockchains, such as Solana, Aptos, and Sui, already employ parallel transaction processing [4, 40, 43, 53, 56]. However, many of these networks have yet to implement fee models that fully account for the challenges of parallel execution. Sui and Solana have introduced fee markets tailored to parallelization, but these mechanisms require users to engage in first-price auctions for congested resources [37, 41]. As a result, these fee markets demand a high level of sophistication from users to effectively optimize their fee settings and are also not incentive-compatible. The requirements for an effective fee market that is suitable for parallel execution and the design of such a market have so far remained unresolved.

Our Contributions. In this work, we aim to bridge this gap by outlining the requirements for such a fee market and evaluating possible candidates. We outline our main contributions below:

- We introduce a framework with two main components: a *Gas Computation Mechanism (GCM)*, which measures the load a transaction imposes on the network in terms of both parallel execution and computation, expressed in *units of gas*, and a Transaction Fee Mechanism (TFM), which determines the cost associated with each unit of gas.
- We introduce a list of desirable properties for a GCM and evaluate against them a set of mechanisms that we propose.
- Our analysis identifies a promising candidate: the *weighted area* GCM. Importantly, this mechanism is not only an effective GCM, achieving a large subset of the outlined properties, but it is equally important that it can be easily integrated with existing TFMs, such as EIP-1559, inheriting their properties.

2 Model

We consider a universe consisting of several stateful *resources* (e.g., user accounts, storage addresses of smart contracts). Each resource can be thought of as a system global variable. We write $\mathcal{R}$ for the set of resources. For analysis purposes, we assume that $\mathcal{R}$ is infinite. A *transaction* is a sequence of elementary instructions performing computation and interacting with the resources. Some of these operations *access* a given target resource (e.g., read its value, write to it). For simplicity, we assume that the following are known in advance and supplied with the transaction:

- The non-empty set of resources $R \subseteq \mathcal{R}$ the transaction accesses, or an overestimate of it.[1]
- The total time $t > 0$ it takes to execute the transaction on a single thread.[2]

Transactions execute concurrently, but *atomically*, meaning that the overall effect of executing a batch of transactions should also be achievable by a sequential, single-core execution. For simplicity, we restrict to *concurrent schedules* following a *simple lock-based execution policy*: each resource has a lock associated with it; whenever a thread wants to execute a transaction, it first locks all required resources, then executes the transaction, and then releases the locks. We assume that acquiring (and similarly releasing) the required locks happens simultaneously and takes no additional time. These simplifications have a desirable side-effect: t and R for each transaction now uniquely determine the set of admissible concurrent schedules, allowing us to ignore other details about the transactions:

Definition 2.1 (Transaction). A transaction tx is specified through a tuple (t, R), where $t > 0$ denotes the time required to execute the transaction and $R \neq \emptyset$ represents the set of resources the transaction demands, which are locked for the duration of the transaction. We will sometimes write $t(tx)$ and $R(tx)$ for t and R respectively.

To give the tuple associated with a transaction, we will write $tx \simeq (t, R)$. Note that different transactions might have the same associated tuple. We will also write $tx_1 \simeq tx_2$ to mean that the two transactions have the same associated tuple.

In Figure 1, we illustrate a sample transaction $tx \simeq (3, \{r_2, r_3\})$. Transaction tx thus takes 3 units of time to execute and utilizes resources r_2 and r_3. We illustrate this by a rectangle of corresponding length (i.e., time) and width (i.e., resources). Throughout, we will illustrate transactions in this manner to aid in visualizing concepts and results. Note that we will always represent transactions as rectangles (i.e., they use consecutive resources). In reality, this is, of course, not the case. Importantly, all our results also hold in the more general setting where a transaction can use any subset of resources.

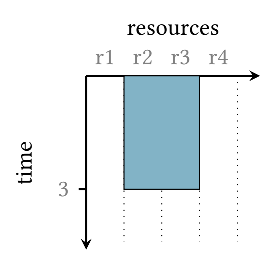

Fig. 1. Illustration of a sample transaction $tx \simeq (3, \{r_2, r_3\})$.

A *blockchain* is a sequence of *blocks* comprising bundles of transactions: $B_1, \ldots, B_k$. The system starts in some predetermined initial state. The blocks are executed in order starting from the oldest (the *genesis* block B_1), successively changing the system's state. For simplicity, we assume no

[1]In Ethereum, transaction accesses are generally not known in advance. Instead, transactions can execute arbitrary logic (constrained by a maximum amount of computational effort) and their execution depends on the blockchain's state at the time of execution. Ethereum currently supports optional *access lists* that allow transactions to specify their accesses [12]. This optional list could be made mandatory to provide an overestimate of accesses. Additionally, it is worth noting that in other blockchains (e.g., Solana [33] and Sui [32]), an overestimate of accesses is typically known in advance.

[2]In reality, the time to execute a transaction depends on the hardware it is executed on. Thus, time is estimated in Ethereum by assigning each operation a computational effort. Then the sum of the computational effort of a transaction's operation can be seen as a proxy for time.

cross-block parallelism, so the execution of a block only starts after the previous block's execution has been completed. However, the execution of transactions inside a block happens concurrently. The system's state after executing B_k is the *current* system state. Users wanting to change the system's state compete for inclusion in the next block B_{k+1} and have to pay a *fee* if successfully included. Desirably, the fee should be higher for more complex transactions and higher during periods of high demand due to limited block space. These requirements are typically decoupled and ensured through different means:

(1) A transaction's complexity is quantified in units of *gas*:[3] the more complex a transaction is, the more gas it consumes. Gas encompasses multiple components such as execution, storage space, and data bandwidth. For our purposes, we will only be concerned with *execution gas*.[4] Currently, the execution gas acts as a proxy for the execution time t,[5] but this need not be the case, as we will demonstrate in our paper.
(2) The fair competition for block space is ensured through a *transaction fee mechanism (TFM)*: users submit transactions they would like to be included in the next block together with bids of what they would be willing to pay per unit of gas. Importantly, block space is limited, i.e., there is limited space for transactions. The mechanism then determines the set of transactions to be included in the block, together with a price per unit of gas to be paid by each included transaction (potentially not the same for all transactions).[6]

As such, an included transaction consuming g gas units at a price of p per unit of gas will have to pay a fee of $g \cdot p$ (generally in the blockchain's native currency).

To keep the separation of concerns in (1) and (2), we will keep the formula for the fee $g \cdot p$ and instead vary the *gas computation mechanism* used to determine g:

Definition 2.2 (Gas Computation Mechanism). A gas computation mechanism (GCM) takes as input a set of transactions T to be included in a block and determines in a deterministic manner the amount of gas consumed by each transaction $tx \in T$, written $\mathrm{gas}_T(tx)$.

Currently deployed GCMs associate a fixed, predetermined gas consumption with each transaction, independent of the specific resources accessed by the transaction, making them unsuitable for a parallel execution environment. In particular, this is true for Ethereum's *current* GCM:

Definition 2.3 (Current GCM). Given a set of transactions T and a transaction $tx \in T$ with $tx \simeq (t, R)$, the *current* GCM computes the amount of gas used by tx as follows:

$$\mathrm{gas}_T^C(tx) := t.$$

Since this does not depend on T, we often drop the subscript.

Our goal is to ensure that fees accurately reflect the parallelizability of transactions. Therefore, the gas consumption calculated for a transaction should depend on the set of resources it accesses and may also be influenced by factors external to the transaction itself (like interactions between transactions).

3 GCM Properties

Next, we outline several desirable properties that a GCM should possess to provide the right incentives for parallelization. These properties should be viewed as a wishlist — as will become clear later on, no single mechanism can satisfy all of them simultaneously.

[3]For Ethereum, the unit is called wei.
[4]From this point forward, gas will refer specifically to execution gas.
[5]Throughout we will assume that execution gas exactly corresponds to time.
[6]There are several other components of a TFM, but for the level of detail we need here, this suffices.

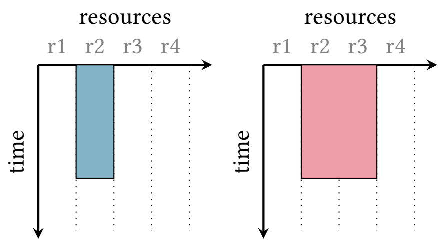

Fig. 2. Illustration of Property 1, where $T = \emptyset$, represents a sample tx_1 and represents a sample tx_2.

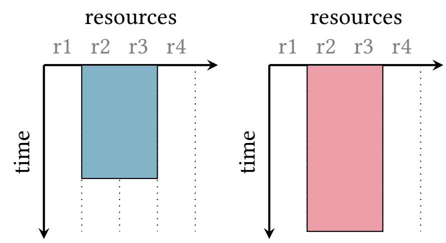

Fig. 3. Illustration of Property 2, where $T = \emptyset$, represents a sample tx_1 and represents a sample tx_2.

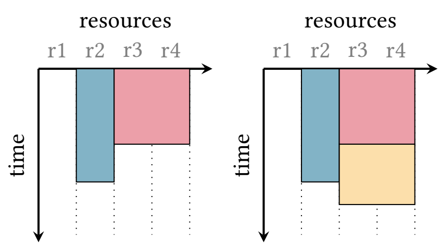

Fig. 4. Illustration of Property 4, where $T = \emptyset$, $T_1 = \{tx_1, tx_2\}$, and $T_2 = T_1 \cup \{tx_3\}$. Here, represents a sample tx_1, represents a sample tx_2, and represents a sample tx_3.

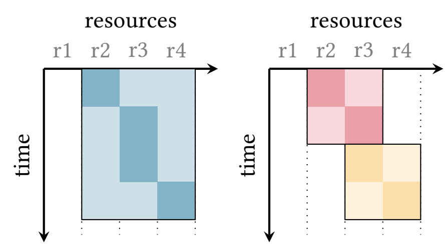

Fig. 5. Illustration of Property 5, where $T = \emptyset$, represents a sample tx_3, while represents a sample tx_1 and represents a sample tx_2. The darker shaded areas indicate when a transaction operates on a resource.

We begin with two natural monotonicity properties, one for resources and one for time. First, a transaction that requires a subset of the resources used by another transaction, while taking the same amount of time, should consume no more gas (Property 1 and Fig. 2). Similarly, a transaction that requires no more execution time than another, assuming both involve the same set of resources, should consume no more gas (Property 2 and Fig. 3).

Property 1 (Resource Sensitivity). *Given a set of transactions T and two transactions $tx_1 \simeq (t_1, R_1)$ and $tx_2 \simeq (t_2, R_2)$, both not in T, such that $t_1 = t_2$ and $R_1 \subseteq R_2$:*

$$gas_{T \cup \{tx_1\}}(tx_1) \leq gas_{T \cup \{tx_2\}}(tx_2).$$

Property 2 (Time Sensitivity). *Given a set of transactions T and two transactions $tx_1 \simeq (t_1, R_1)$ and $tx_2 \simeq (t_2, R_2)$, both not in T, such that $t_1 \leq t_2$ and $R_1 = R_2$:*

$$gas_{T \cup \{tx_1\}}(tx_1) \leq gas_{T \cup \{tx_2\}}(tx_2).$$

The previous two properties fix one dimension while varying the other. One can also define a seemingly stronger property that allows both to vary, as follows:

Property 3 (Resource-Time Sensitivity). *Given a set of transactions T and two transactions $tx_1 \simeq (t_1, R_1)$ and $tx_2 \simeq (t_2, R_2)$, both not in T, such that $t_1 \leq t_2$ and $R_1 \subseteq R_2$:*

$$gas_{T \cup \{tx_1\}}(tx_1) \leq gas_{T \cup \{tx_2\}}(tx_2).$$

Intuitively, if $tx_1 \lesssim tx_2$, by which we mean $t(tx_1) \leq t(tx_2)$ and $R(tx_1) \subseteq R(tx_2)$, then tx_1 should cost no less than tx_2. However, an attentive reader will observe that the former two are collectively equivalent to the latter (the proof and all subsequent omitted proofs can be found in the appendix):

LEMMA 3.1. *Property 3 holds if and only if Properties 1 and 2 hold.*

Let us now take a moment to briefly evaluate why Properties 1 to 3 are not merely intuitive, but their violation can lead to harmful consequences and misaligned incentives: suppose $tx_1 \lesssim tx_2$ and T is a set of transactions containing neither of the two. If $\text{gas}_{T\cup\{tx_1\}}(tx_1) > \text{gas}_{T\cup\{tx_2\}}(tx_2)$, a user intending to submit tx_1 might instead pad tx_1 with unnecessary instructions and declare a larger access list to decrease the gas consumption. This would paradoxically reduce the gas usage and, assuming a reasonable TFM is used to compute transaction fees, also lower the transaction fee.

For any of the three properties above, we say that the property is *strictly* satisfied if for $tx_1 \not\simeq tx_2$ the conclusion inequality holds strictly. Note that properties holding strictly are even more desirable with respect to the reasoning above: replacing a transaction with a "larger" one is then not only no better but actively worse. Unsurprisingly, Lemma 3.1 also holds for the strict versions of the properties:

LEMMA 3.2. *Property 3 holds strictly if and only if Properties 1 and 2 hold strictly.*

Next, we introduce another desirable monotonicity property, this time with a different emphasis: if two sets of transactions satisfy $T_1 \subseteq T_2$, the transactions in T_1 should collectively consume no more gas than those in T_2 (Property 4 and Fig. 4). To formalize this, given a set of transactions T and a subset $T' \subseteq T$, write $\text{gas}_T(T') := \sum_{tx' \in T'} \text{gas}_T(tx')$ for the total gas consumed by the transactions in T' when included in the set T that constitutes a block.

PROPERTY 4 (SET INCLUSION). *Given a set of transactions T and two sets of transactions $T_1 \subseteq T_2$, disjoint from T:*

$$gas_{T\cup T_1}(T_1) \leq gas_{T\cup T_2}(T_2).$$

To understand why this property is desirable, consider a GCM for which it does not hold. Then, there must be a scenario where a set of transactions can reduce their total gas consumption by adding additional transactions. This situation could be exploited through collusion by the users originating these transactions. Importantly, such a possibility is undesirable, as it would primarily benefit sophisticated users capable of orchestrating such arrangements.

Similarly to before, we say that Property 4 holds *strictly* if for $T_1 \neq T_2$ the inequality in the conclusion holds strictly, which is desirable for reasons similar to those discussed above.

We now move on to more complex properties. Given two transactions $tx_1 \simeq (t_1, R_1)$ and $tx_2 \simeq (t_2, R_2)$, a transaction tx_3 is their *sequential composition* (or, more simply, *concatenation*), if it executes the steps of tx_1 followed by the steps of tx_2. Note that, in this case $tx_3 \simeq (t_1 + t_2, R_1 \cup R_2)$. Our next property states that the concatenation of two transactions should consume no less gas than submitting them individually (Property 5 and Fig. 5). The two individual transactions perform the same actions as their concatenation, but not atomically, with no guarantee of their relative ordering or control over what happens between them. Naturally, enforcing atomicity and ordering limits the set of admissible concurrent schedules and requires resources to remain locked over longer continuous timespans. In particular, tx_3 requires the resources in $R_1 \cup R_2$ to be locked over a continuous span of t_1+t_2 units, while tx_1 and tx_2 submitted individually only require exclusive access to resources in R_1 for t_1 units and to resources in R_2 for t_2 units. Hence, the "bigger" transaction is at least as hard to schedule as its two constituent "parts" and should hence consume no less gas.

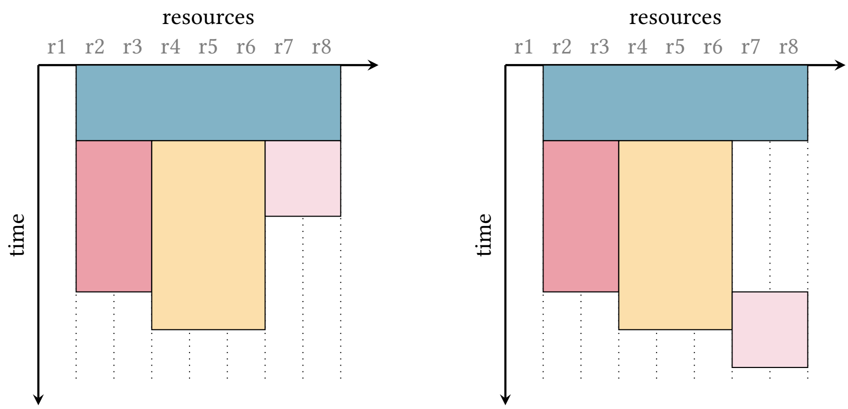

Fig. 6. Illustration of optimal schedules for a set of four transactions: $tx_1 \simeq (2, \{r_2, \ldots, r_8\})$ (shown in), $tx_2 \simeq (4, \{r_2, r_3\})$ (shown in), $tx_3 \simeq (5, \{r_4, r_5, r_6\})$ (shown in), and $tx_4 \simeq (2, \{r_7, r_8\})$ (shown in). In the left plot, we show the optimal schedule for $n = 3$, and in the right plot for $n = 2$. Notice how for $n = 2$, we cannot schedule the three transactions tx_2, tx_3, and tx_4 to be executed in parallel even though they access pairwise-disjoint sets of resources.

Property 5 (Transaction Bundling). *Consider a set of transactions T and three transactions $tx_1, tx_2, tx_3 \notin T$ such that tx_3 is the concatenation of tx_1 and tx_2, then:*

$$gas_{T \cup \{tx_1, tx_2\}}(tx_1) + gas_{T \cup \{tx_1, tx_2\}}(tx_2) \leq gas_{T \cup \{tx_3\}}(tx_3).$$

Let us again consider the risks of having a GCM that does not satisfy the previous property. In such a scenario, a group of users could collude to collectively consume less gas by combining their transactions into a single transaction rather than processing them individually. This outcome would be undesirable, particularly because it disproportionately benefits sophisticated users, as we outlined before. Note that this could even be the case for a single user wanting to submit multiple transactions.

We say that Property 5 holds *strictly* if the inequality in the conclusion holds strictly, which is again more desirable than the basic version of the property.

Next, we would like to formalize the intuitive idea that a transaction's gas consumption fairly reflects its impact on the execution time. To do so, we first need to formalize scheduling more precisely. Let $n \geq 2$ be the number of available threads.

Definition 3.3. A *scheduler* (for n threads) takes as input a set of transactions T, and outputs a *concurrent schedule* using at most n threads to execute all transactions in T. This schedule specifies the operations each thread should perform and the order in which they should be performed.

The following conditions should hold for any generated schedule:

- Each transaction is assigned to a single thread;
- A thread can work on only one transaction at a time;
- There is no preemption: once a thread starts executing a transaction, it completes the transaction without context switching;
- Transactions with overlapping resource access sets cannot be executed in parallel: one must finish before the other can begin.

Note that some of these conditions are natural for scheduling in general, while others arise from us assuming the *simple lock-based execution policy.*[7] For our purposes, we are not concerned with which thread executes which transaction, but only that no more than n transactions ever execute simultaneously (so we can draw concurrent schedules as in Fig. 6, which illustrates the concept). To determine gas consumption, our notation will need to capture even less — given a scheduler s (for n threads), we write $v^s(T)$ for the *makespan* of the schedule produced by s for the set of transactions T (i.e., the time required to execute the schedule in parallel using n threads). It is instructive to read the paper having in mind as a prime example s being the *optimal scheduler* (for n threads), which returns a schedule for n threads that minimizes the makespan. However, implementing such a scheduler is computationally prohibitive in practice, so greedy heuristics are typically used instead. To make our results apply even to non-optimal schedulers, we assume a set of minimal, reasonable properties for the scheduler:

(S1) *Monotonicity in T:* for any $T \subseteq T'$, we have $v^s(T) \leq v^s(T')$. Intuitively, scheduling no fewer transactions takes no less time.
(S2) *Monotonicity under bundling:* consider any set of transactions T and three transactions $tx_1, tx_2, tx_3 \notin T$, such that tx_3 is the concatenation of tx_1 and tx_2, then we have $v^s(T \cup \{tx_1, tx_2\}) \leq v^s(T \cup \{tx_3\})$. Intuitively, replacing two transactions by their concatenation makes scheduling no easier.
(S3) *Monotonicity in t and R:* consider any set of transactions T and two transactions $tx_1 \simeq (t_1, R_1)$ and $tx_2 \simeq (t_2, R_2)$, both not in T, such that $t_1 \leq t_2$ and $R_1 \subseteq R_2$, i.e., $tx_1 \lesssim tx_2$, then we have $v(T \cup \{tx_1\}) \leq v(T \cup \{tx_2\})$. Intuitively, "larger" transactions are no easier to schedule.
(S4) *Empty set:* $v(\emptyset) = 0$.

The optimal scheduler can be easily seen to satisfy these properties.[8] *Our positive results will apply to any scheduler satisfying the properties, while our negative results will be for the optimal scheduler itself.* Henceforth, we drop the s superscript for brevity and follow this convention for who s is.

An attentive reader may notice that (S1)–(S3) partially resemble Properties 1 to 5 for GCMs. This resemblance is not coincidental, but it is important to emphasize that they address different aspects: the former are properties of the scheduler, while the latter are properties of the GCM. A GCM can, in fact, satisfy Properties 1 to 5 without depending on the scheduler at all (e.g., the current mechanism). Conversely, for mechanisms defined in terms of the scheduler, (S1)–(S3) play a crucial role in establishing their properties, including Properties 1 to 5.

Armed as such, we now return to our latest goal: formalizing the idea that a transaction's gas consumption should fairly reflect its impact on execution time. We do this as follows:

Property 6 (Scheduling Sensitivity). *Given a set of transactions T and two transactions tx_1 and tx_2, both not in T, such that $v(T \cup \{tx_1\}) < v(T \cup \{tx_2\})$:*[9]

$$gas_{T \cup \{tx_1\}}(tx_1) \leq gas_{T \cup \{tx_2\}}(tx_2).$$

[7]Using locks is one way to enforce this policy, but in our case — where the contents of a block are known before execution commences — it can also be achieved without locks, as long as the execution environment ensures that threads strictly follow a pre-determined schedule. We chose this name because it is largely suggestive of the intended semantics.

[8]All but the second straightforward, while for the second, any admissible schedule for $T \cup \{tx_3\}$ can be turned into a same-makespan admissible schedule for $T \cup \{tx_1, tx_2\}$ by replacing tx_3 by tx_1 followed immediately by tx_2.

[9]Interestingly, unlike our earlier properties, writing $v(T \cup \{tx_1\}) \leq v(T \cup \{tx_2\})$ here may be too strong, as applying the property twice would then imply that if $v(T \cup \{tx_1\}) = v(T \cup \{tx_2\})$, then $\mathrm{gas}_{T \cup \{tx_1\}}(tx_1) = \mathrm{gas}_{T \cup \{tx_2\}}(tx_2)$, which is not necessarily desirable. Writing $v(T \cup \{tx_1\}) \leq v(T \cup \{tx_2\})$ would also unintentionally make the strict and non-strict versions of the property incomparable.

Intuitively, transactions with higher marginal contributions to the execution time should consume no less gas.[10] As usual, we define a *strict* version of this property, where the latter inequality also becomes strict (i.e., higher marginal contributions in execution time imply higher gas consumption). One might be tempted to believe that Scheduling Sensitivity implies Resource-Time Sensitivity. However, proving this requires a non-strict inequality in the premise $v(T \cup \{tx_1\}) < v(T \cup \{tx_2\})$.

There is a second way in which gas consumption should adequately indicate the effort required to execute transactions (Property 7). Specifically, the gas consumption of all transactions in a block should *collectively* account for the total time needed to execute the block. This can be seen as properly tracking the time consumed by the execution environment to execute the block.

Property 7 (Efficiency). *Consider a set of transactions T and recall the definition $gas_T(T) = \sum_{tx \in T} gas_T(tx)$, then:*

$$gas_T(T) = v(T).$$

Next, we introduce two practical properties. The first requires that transaction submitters be able to estimate a transaction's gas consumption in advance. We formalize this by requiring that $\text{gas}_T(tx)$ does not depend on T (Property 8).

Property 8 (Easy Gas Estimation). *Given two sets of transactions T_1 and T_2 and a transaction tx belonging to neither T_1 nor T_2:*

$$gas_{T_1 \cup \{tx\}}(tx) = gas_{T_2 \cup \{tx\}}(tx).$$

This property ensures a good user experience by making it easy for users to estimate gas usage. If the gas consumption of a transaction depended on the remaining transactions in the block, this estimation would become significantly more complex. In general, we aim to keep gas estimation straightforward to avoid giving an advantage to more sophisticated users. Additionally, as we will see later, a GCM satisfying this property can be seamlessly composed with existing TFMs, retaining their desirable properties (see Section 6). Sadly, as one might have already guessed, Easy Gas Estimation is incompatible with Scheduling Sensitivity (except for *constant* mechanisms, i.e., GCMs that return a constant independent of T and tx), and also incompatible with Efficiency:

Theorem 3.4. *Easy Gas Estimation (Property 8) is:*

(1) Incompatible with Scheduling Sensitivity (Property 6) unless using a constant GCM.[11]
(2) Incompatible with Efficiency (Property 7).

Proof. Consider a mechanism M with Easy Gas Estimation; i.e., $\text{gas}^M(tx)$ is meaningful without a subscript for the set of transactions T in the block. Then:

(1) Assume that M has Scheduling Sensitivity; we will show that M is a constant mechanism.

Call two transactions $tx_1 \simeq (t_1, R_1)$ and $tx_2 \simeq (t_2, R_2)$ *incomparable* if neither $R_1 \subseteq R_2$ nor $R_2 \subseteq R_1$. As a first step, we will show that if tx_1 and tx_2 are incomparable, then $\text{gas}^M(tx_1) = \text{gas}^M(tx_2)$. To show this, assume that $R_1 \not\subseteq R_2$. We will show that then $\text{gas}^M(tx_1) \geq \text{gas}^M(tx_2)$. Exchanging the roles of tx_1 and tx_2 will then give the conclusion. Let $r \in R_1 \setminus R_2$ and tx_3 be another transaction such that $tx_3 \simeq (t_3, \{r\})$ for some $t_3 > t_2$, and define $T = \{tx_3\}$. Then, for any number of threads $n \geq 2$ and any scheduler that is optimal for two transactions, we have $v(T \cup \{tx_1\}) = t_1 + t_3 > t_3 = v(T \cup \{tx_2\})$. By Scheduling Sensitivity, this implies that $\text{gas}^M(tx_1) \geq \text{gas}^M(tx_2)$.

[10]Technically, what we wrote above in Property 6 are not marginal contributions, but can be made to be by subtracting $v(T)$ from both sides of the inequality in the antecedent.
[11]Constant GCMs satisfy Scheduling Sensitivity, but not Strict Scheduling Sensitivity.

We now know that M associates the same gas consumption to any pair of incomparable transactions. Armed as such, consider two arbitrary transactions $tx_1 \simeq (t_1, R_1)$ and $tx_2 \simeq (t_2, R_2)$. Let r be an arbitrary resource *not* in $R_1 \cup R_2$ and tx_3 be a transaction such that $tx_3 \simeq (1, \{r\})$. Since R_1 and R_2 are non-empty, one can easily see that tx_3 is incomparable with both tx_1 and tx_2, from which $\text{gas}^M(tx_1) = \text{gas}^M(tx_3) = \text{gas}^M(tx_2)$.

(2) Assume for a contradiction that M has Efficiency, then for any transaction $tx \simeq (t, R)$ we have $\text{gas}^M(tx) = \text{gas}^M_\varnothing(\{tx\}) = v(\{tx\}) = t$. Hence, by definition, M is the current mechanism, which can be easily seen to not satisfy Efficiency: assume $n \geq 2$ threads and consider the set of transactions $T = \{tx_1, tx_2\}$ where $tx_1 \simeq (1, \{r_1\})$ and $tx_2 \simeq (1, \{r_2\})$. In this case, $\text{gas}^M(T) = \text{gas}^M(tx_1) + \text{gas}^M(tx_1) = 1 + 1 = 2 \neq 1 = v(T)$. □

Our second practical property requires that gas consumption be efficiently computable, i.e., in polynomial time (Property 9). A GCM that does not satisfy this property would be unsuitable for the blockchain context, where gas computation is intended to be a straightforward component.

Property 9 (Poly-time Computable). *There exists a polynomial-time algorithm that takes as input a transaction set T and a transaction tx and outputs $\text{gas}_{T\cup\{tx\}}(tx)$.*

4 GCM Proposals

We now propose multiple GCM designs.[12] Motivated by the incompatibilities in Theorem 3.4, these fall into two categories:

(C1) Mechanisms with Easy Gas Estimation, in which each transaction's gas consumption is computed in isolation. Such mechanisms tend to be both straightforward and attractive but can achieve neither Scheduling Sensitivity[13] nor Efficiency.

(C2) Mechanisms *without* Easy Gas Estimation, which, given a set of transactions T, rely on $v(T)$ or, more generally, $(v(T'))_{T'\subseteq T'}$ to holistically calculate gas consumptions for the block T, requiring knowledge of the entire block. Instead, these mechanisms aim to achieve Scheduling Sensitivity and/or Efficiency.

4.1 Mechanisms with Easy Gas Estimation

We already introduced the *current* GCM (Definition 2.3). For the next mechanism, we assume a globally available vector of *positive* weights for the resources $(w_r)_{r\in\mathcal{R}}$. For instance, these weights could all be 1. Alternatively, higher-weight resources could correspond to resources under higher (historical) demand. For our purposes, we only need to assume that the weights are known and do not depend on the block being built. In practice, one could update the weights between blocks to accurately reflect resource demand (the exact details are not relevant here; see Section 6 for further discussion). Given the weights, we define:

Definition 4.1 (Weighted Area GCM). Given a set of transactions T and a transaction $tx \in T$ with $tx \simeq (t, R)$, the *Weighted Area* GCM computes the amount of gas used by tx as follows:

$$\text{gas}^{WA}_T(tx) := t \cdot \left(1 + \sum_{r\in R} w_r\right) \tag{1}$$

Since this does not depend on T, we often drop the subscript.

[12]Note again that we will solely consider deterministic mechanisms. This is common for blockchain fee markets given the difficulty of having a true source of randomness on blockchains.
[13]Except for constant GCMs.

To gain intuition, it is instructive to consider the former case, where all weights are 1, i.e., the *unweighted area* mechanism, in which case Eq. (1) becomes $\text{gas}_T^{WA}(tx) = t \cdot (1 + |R|)$. This mechanism can also be viewed as the addition of two terms: the *current term*, i.e., the gas consumption of *tx* under the *current* GCM, namely t, and the *area term*, i.e., the area of a $t \times |R|$ rectangle (which is also how we draw transactions in our diagrams). The *current term* is helpful to ensure no transaction ever costs too little when the weights of all resources accessed by a transaction are too close to zero.[14]

The *area term*, on the other hand, which is the main component, intuitively corresponds to "negated throughput." That is, executing the transaction requires holding $|R|$ locks for t time units, during which no other transactions requiring any of those resources can execute. In Fig. 7, we illustrate this idea. Transaction *tx* prevents the execution of other transactions across its entire resource set but only utilizes each resource for a short period. While the current GCM charges a transaction based solely on its total computation time (i.e., the height of the rectangle), the weighted area GCM also accounts for the resources for which it "negated throughput", i.e., the area occupied by the transaction. This occupied area prevents other transactions using the same resources from being scheduled in parallel.

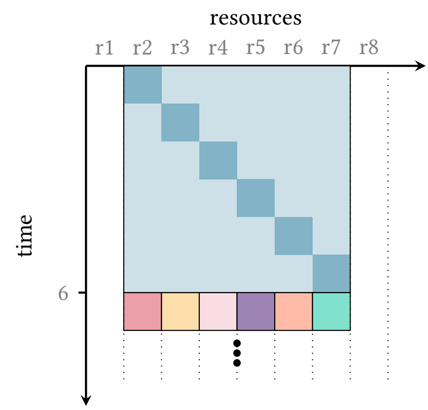

Fig. 7. Illustration of a worst-case transaction in terms of parallelizability. Transaction $tx \simeq (6, \{r_2, \ldots, r_7\})$ (shown in ☐) takes 6 units of time to execute and accesses 6 resources. However, the transaction only operates on each resource for one unit of time each (shown in ■).

Along the same line of reasoning, to further reinforce why this mechanism is a reasonable choice, consider a transaction set T. The sum $\sum_{tx \in T} t(tx) \cdot |R(tx)|$ represents the "total area" of transactions in T. Dividing this term by the number of resources used in T provides a lower bound on $v(T)$, serving as a rough proxy that does not require knowledge of T when computing individual gas consumptions. Finally, we note that, even in the case of non-unit weights, the term *weighted area* remains meaningful, as the area term still corresponds to the area of the respective rectangle in our diagrams if we give the column of each resource $r \in \mathcal{R}$ a width of w_r.

4.2 Mechanisms *without* Easy Gas Estimation

We now switch gears towards mechanisms without Easy Gas Estimation that instead aim to achieve Scheduling Sensitivity and/or Efficiency. Intuitively, for a set of transactions T, such mechanisms should start from $v(T)$ and take each transaction's marginal contribution towards $v(T)$ as its gas consumption. This, however, has to be done carefully, as, e.g., simply looking for each transaction $tx \in T$ at $v(T) - v(T \setminus \{tx\})$ does not help. Note that one can easily construct cases where removing any one transaction from T does not change $v(T)$, so all reported numbers will be 0. However, all these numbers being zero does not mean that no transaction contributes to $v(T)$; it just means that we also have to consider removing multiple transactions to see the differences. This gives us the idea to look at the marginal contribution of each transaction $tx \in T$ when added to each possible

[14]This term has a practical motivation. In reality, transactions will be scheduled on a small finite number of threads. Thus, even if they do not access any high-demand resource, they occupy space in the schedule proportional to their execution time. Note, moreover, that this term could technically be omitted, and our results would still hold.

subset $S \subseteq T \setminus \{tx\}$, and not just to $S = T \setminus \{tx\}$ as before. Formally, we would like to compute tx's gas consumption as an aggregate of $(v(S \cup \{tx\}) - v(S))_{S \subseteq T \setminus \{tx\}}$. We next present two mechanisms based on this idea: the *Shapley* (Definition 4.2) and *Banzhaf* (Definition 4.4) GCMs, inspired by corresponding concepts in cooperative game theory.

Definition 4.2 (Shapley GCM). Given a set of transactions T consisting of $|T| = n$ transactions and a transaction $tx \in T$, the *Shapley* GCM computes the amount of gas used by tx as follows:

$$\text{gas}_T^S(tx) := \frac{1}{n!} \sum_{\sigma} \left[v(P_{tx}^{\sigma} \cup \{tx\}) - v(P_{tx}^{\sigma}) \right] \tag{2}$$

$$= \sum_{S \subseteq T \setminus \{tx\}} \frac{|S|! \cdot (n - |S| - 1)!}{n!} \left[v(S \cup \{tx\}) - v(S) \right]. \tag{3}$$

Here, σ ranges over the $n!$ possible ways to order the transactions in T and P_{tx}^{σ} denotes the set of transactions that precede tx in the order σ. The equality between Eqs. (2) and (3) follows by counting the number of orders σ such that $P_{tx}^{\sigma} = S$, which there are $|S|! \cdot (n - |S| - 1)!$ of.

Another way to understand the Shapley GCM is through the following probabilistic experiment:

(1) Select an ordering σ of the transactions in T uniformly at random.
(2) Start with an empty set of transactions and add transactions one by one in the order given by σ.
(3) Whenever a transaction tx is added, let $S = P_{tx}^{\sigma}$ be the set of transactions just before adding it and compute tx's *marginal contribution* to the execution time as $v(S \cup \{tx\}) - v(S)$; i.e., by how much did the execution time increase by adding tx to the current set of transactions.
(4) The gas consumption of $tx \in T$ is the expectation of its marginal contribution across the experiment.

The reader familiar with cooperative game theory will have already recognized the immediate connection with Shapley values: if we see each transaction $tx \in T$ as a player in a game with valuation function $v : 2^T \to \mathbb{R}$, then $\text{gas}_T^S(tx)$ is precisely the celebrated *Shapley value* of player tx, more traditionally written $\phi_{tx}(v)$. One classical property of Shapley values is that, given $v(\emptyset) = 0$, as is the case for us by (S4), their sum equals the valuation of the *grand coalition* T: $\sum_{tx \in T} \phi_{tx}(v) = v(T)$. The proof is straightforward: for any fixed ordering σ of T, the sum of the marginal contributions of the transactions is a telescoping sum, i.e., except first and last terms, all others appear once positively and once negatively, canceling as a result and leaving us with $v(T) - v(\emptyset) = v(T)$. Since the sum does not depend on σ, the same holds when taking the expectation with respect to σ. This already gives us our first property of the Shapley GCM, namely Efficiency:

Lemma 4.3. *The Shapley GCM satisfies Efficiency (Property 7).*

We postpone studying the Shapley GCM further until introducing our other mechanisms in the section.

A second GCM based on the idea that a transaction's gas consumption should be its marginal contribution to the execution time is the *Banzhaf* mechanism:

Definition 4.4 (Banzhaf GCM). Given a set of transactions T consisting of $|T| = n$ transactions and a transaction $tx \in T$, the *Banzhaf* GCM computes the amount of gas used by tx as follows:

$$\text{gas}_T^B(tx) := \frac{1}{2^{n-1}} \sum_{S \subseteq T \setminus \{tx\}} \left[v(S \cup \{tx\}) - v(S) \right].$$

As for the Shapley mechanism, the Banzhaf mechanism can be understood probabilistically, but this time with a separate experiment for each transaction tx. Sample a subset S of transactions other than tx uniformly at random and compute tx's marginal contribution to the execution time when added to S, namely $v(S \cup \{tx\}) - v(S)$. The gas consumption of tx is then its expected marginal contribution to the execution time. The familiar reader will recognize this as the definition of the *Banzhaf power index* $\beta_{tx}(v)$. For consistency with *Shapley values*, we will instead call these *Banzhaf values*. While more straightforward than the corresponding experiment used in defining Shapley values, the fact that we now have n separate experiments makes the sum of the values no longer well-behaved, losing Efficiency for the Banzhaf mechanism:

LEMMA 4.5. *The Banzhaf GCM does* not *satisfy Efficiency (Property 7).*

We conclude this section by introducing two additional reasonable mechanisms *without* Easy Gas Estimation. These mechanisms are notably more straightforward than the Shapley and Banzhaf GCMs, as they avoid computing marginal contributions. However, they have other drawbacks that will become more apparent when we begin studying their normative properties alongside the other mechanisms.

Definition 4.6 (Time-Proportional Makespan GCM). Given a set of transactions T and a transaction $tx \in T$, the *Time-Proportional Makespan (TPM)* GCM computes the amount of gas used by tx as follows:

$$\text{gas}_T^{TPM}(tx) := \frac{t(tx)}{\sum_{tx' \in T} t(tx')} \cdot v(T).$$

Definition 4.7 (Equally-Split Makespan GCM). Given a set of transactions T and a transaction $tx \in T$, the *Equally-Split Makespan (ESM)* GCM computes the amount of gas used by tx as follows:

$$\text{gas}_T^{ESM}(tx) := \frac{v(T)}{|T|}.$$

We also consider the following rather pathological mechanism because of the combination of properties it turns out to satisfy (that none of our other mechanisms do):

Definition 4.8 (Exponentially-Split Makespan GCM). Given a set of transactions T and a transaction $tx \in T$, the *Exponentially-Split Makespan (XSM)* GCM computes the amount of gas used by tx as follows:

$$\text{gas}_T^{XSM}(tx) := \frac{v(T)}{3^{|T|}}.$$

5 Analysis of Our GCMs

In this section, we provide a detailed analysis of the normative properties of our GCMs. Our results are summarized in Table 1.

5.1 Mechanisms with Easy Gas Estimation

In this section, we analyze the current and Weighted Area GCMs. By definition, both satisfy Easy Gas Estimation (Property 8) and are Poly-time Computable (Property 9). Since they satisfy Easy Gas Estimation but are not constant GCMs, Theorem 3.4 implies that *neither* satisfies Scheduling Sensitivity (Property 6) nor Efficiency (Property 7). Furthermore, both mechanisms satisfy strict Time Sensitivity (Property 2): increasing the time t of a transaction strictly increases its gas consumption. Similarly, both satisfy strict Set Inclusion (Property 4): since transactions' gas consumptions are

Property	Current	W. Area	Shapley	Banzhaf	TPM	ESM	XSM
Resource Sensitivity (Property 1)	$=$	$<$	$\leq$	$\leq$	$\leq$	$\leq$	$\leq$
Time Sensitivity (Property 2)	$<$	$<$	$<$	$<$	$<$	$\leq$	$\leq$
Resource-Time Sensitivity (Property 3)	$\leq$	$<$	$\leq$	$\leq$	$\leq$	$\leq$	$\leq$
Set Inclusion (Property 4)	$<$	$<$	×	×	$\leq$	$\leq$	×
Transaction Bundling (Property 5)	$=$	$\leq$	×	$\leq$	$\leq$	×	$<$
Scheduling Sensitivity (Property 6)	×	×	×	×	×	$<$	$<$
Efficiency (Property 7)	×	×	✓	×	✓	✓	×
Easy Gas Estimation (Property 8)	✓	✓	×	×	×	×	×
Poly-time Computable (Property 9)	✓	✓	$S(v)$	$B(v)$	v	v	v

Table 1. Comparison of GCMs based on their adherence to the defined properties. $<$ indicates that the mechanism *strictly* satisfies the property, $=$ indicates *trivial* satisfaction (by equality), and $\leq$ indicates satisfaction (not necessarily strict). A "✓" means the property is satisfied; "×" means it is not satisfied. For computational complexity, "v" means the mechanism is as hard to compute as $v(\cdot)$ itself, "$S(v)$" means it is as hard as computing Shapley values for v, and "$B(v)$" means it is as hard as computing Banzhaf values for v.

strictly positive, a strict superset of a given set of transactions consumes strictly more gas. For the remaining three properties, the two mechanisms behave slightly differently:

Resource Sensitivity (Property 1). The current mechanism ignores the set of resources R that a transaction accesses, so replacing R with a strict superset of it does not change the transaction's gas consumption. Therefore, the current mechanism satisfies Resource Sensitivity *with equality*. On the other hand, the Weighted Area mechanism adds an extra term of $t \cdot w_r > 0$ to the gas consumption for each additional resource r added to R, so it satisfies strict Resource Sensitivity.

Resource-Time Sensitivity (Property 3). From the above results on whether our two mechanisms satisfy Resource Sensitivity and Time Sensitivity, Lemmas 3.1 and 3.2 allow us to conclude that the current mechanism satisfies Resource-Time Sensitivity, while the Weighted Area mechanism satisfies it strictly.

Transaction Bundling (Property 5). Concatenating two transactions with times t_1 and t_2 results in a transaction with time $t_1 + t_2$. Since the current mechanism equates gas consumption with time, the bundled transaction has the same gas consumption as the two individual transactions combined. This implies that the current mechanism satisfies Transaction Bundling *with equality*. Finally, the Weighted Area mechanism satisfies Transaction Bundling, as shown below. If we restrict ourselves to bundling transactions with different resource sets, the property is strictly satisfied.

LEMMA 5.1. *The Weighted Area GCM satisfies Transaction Bundling (Property 5). If we only consider bundling transactions with different resource sets, the property is strictly satisfied.*

PROOF. Consider two transactions $tx_1 \simeq (t_1, R_1)$ and $tx_2 \simeq (t_2, R_2)$. Let $tx_3 \simeq (t_1 + t_2, R_1 \cup R_2)$ be a transaction consisting of the concatenation of tx_1 and tx_2. We want to show that $\text{gas}^{WA}(tx_1) + \text{gas}^{WA}(tx_2) \leq \text{gas}^{WA}(tx_3)$. By definition, this amounts to:

$$t_1 \cdot \left(1 + \sum_{r \in R_1} w_r\right) + t_2 \cdot \left(1 + \sum_{r \in R_2} w_r\right) \leq (t_1 + t_2) \cdot \left(1 + \sum_{r \in R_1 \cup R_2} w_r\right) \tag{4}$$

Which is true because $\sum_{r \in R_i} w_r \leq \sum_{r \in R_1 \cup R_2} w_r$ for $i \in \{1, 2\}$. Because the weights are strictly positive, equality occurs if and only if $R_1 = R_1 \cup R_2$ and $R_2 = R_1 \cup R_2$; i.e., $R_1 = R_2$. Hence, the property is satisfied strictly if we restrict bundling transactions to cases where $R_1 \neq R_2$. □

5.2 Mechanisms *without* Easy Gas Estimation

In this section, we analyze the Shapley, Banzhaf, TPM, ESM, and XSM GCMs. By definition, all of them require knowledge of T to compute $\text{gas}_T(tx)$, so they do not satisfy Easy Gas Estimation (Property 8). Next, we examine each of the remaining properties individually and analyze whether they hold for our five mechanisms.

Poly-time Computable (Property 9). The TPM, ESM, and XSM GCMs are all Poly-time Computable whenever determining the makespan of a given set of transactions under the chosen scheduler is feasible in polynomial time, i.e., when $v(T)$ can be computed in polynomial time.[15] In fact, the problems are all equally difficult to computing such makespans. In contrast, computing gas consumptions under the Shapley and Banzhaf GCMs hits another hurdle: it is no longer enough to be able to efficiently compute $v(T)$, but instead, one needs to be able to compute the Shapley or Banzhaf values of the transactions, involving aggregating over $(v(T'))_{T' \subseteq T}$. Hence, computing Shapley and Banzhaf values is, in general, NP-hard and #P-complete [17, 22, 38, 39, 48]. Moreover, no deterministic polynomial-time algorithm can approximate the Shapley values within a constant factor unless P = NP. Using randomization could, in principle, circumvent this: to approximate an average consisting of exponentially many terms, sample polynomially many uniformly at random, and take their average. However, randomization would not be appropriate for the blockchain context, and this still runs into the problem of computing the sampled terms, which is as hard as evaluating v a constant number of times. See [13] for an ampler discussion of computing Shapley and Banzhaf values. Note that the previous results mostly pertain to functions v of a different shape than ours (i.e., not related to scheduling). We have not attempted to show that hardness is retained in our context, but expect this to be true. Note still that because the Shapley GCM is Efficient, it can be used to compute $v(T)$ by adding up the gas consumptions of all transactions in T, meaning that the Shapley GCM is at least as difficult to compute as $v(T)$. However, this simple reduction no longer works for the Banzhaf GCM.

Efficiency (Property 7). We have already seen that the Shapley GCM is Efficient (Lemma 4.3) while the Banzhaf GCM is not (Lemma 4.5). Moreover, by adding up the gas consumption, one can immediately see from the definitions that TPM and ESM are Efficient, while XSM is not.

Scheduling Sensitivity (Property 6). The ESM and XSM GCMs can be easily seen to strictly satisfy Scheduling Sensitivity. This is because, under both mechanisms, given a set of transactions T, the gas consumption of any transaction in T is computed as $f(|T|) \cdot v(T)$, where f is either $\frac{1}{x}$ or $\frac{1}{3^x}$. Notably, the first factor depends only on $|T|$, so if a transaction in T were modified in a way that increases the makespan, this increase would also be reflected proportionally in its gas consumption. In contrast, the Shapley, Banzhaf, and TPM GCMs do *not* satisfy Scheduling Sensitivity (proven in Lemma 5.2 below). This may be particularly surprising for the Shapley and Banzhaf GCMs, as they were specifically designed to account for marginal increases in makespan. The catch is that Scheduling Sensitivity considers replacing a transaction tx with another one leading to an increase in makespan. However, some elements in $(v(S \cup \{tx\}) - v(S))_{S \subseteq T \setminus \{tx\}}$ may still decrease as a result (except for $S = T \setminus \{tx\}$, for which we assumed an increase). Because both the Shapley and Banzhaf values of tx take a weighted average over these values, the average might still decrease, which is what happens in the example in the lemma below.

[15] This is essentially never true for non-trivial makespan minimization problems. In particular, if we restrict our attention to the case of unit-length transactions (i.e., with $t = 1$) and infinitely many threads $n = \infty$, then checking for the existence of a schedule with makespan c corresponds to checking whether the intersection graph of the transactions (i.e., where edges correspond to transactions with intersecting resource sets) is c-colorable. For $c = 3$, this is well-known to be NP-complete, but this result is for general graphs. However, there is a straightforward way to model any graph $G = (V, E)$ as an intersection graph of transactions: vertices $v \in V$ are transactions and for every edge $(u, v) \in E$, create a new resource r and add it to the resource sets of transactions u and v.

Lemma 5.2. *The Shapley, Banzhaf and TPM GCMs do* not *satisfy Scheduling Sensitivity (Property 6).*

Proof. Consider the set of transactions $T = \{tx_1, tx_2\}$ and two additional transactions tx_3 and tx_4 such that $tx_1 \simeq (1, \{r_1\})$, $tx_2 \simeq (3, \{r_2\})$, $tx_3 \simeq (2, \{r_1\})$, and $tx_4 \simeq (1, \{r_2\})$. Then, for any number of threads $n \geq 2$ and the optimal scheduler we have $v(T \cup \{tx_3\}) = 3 < 4 = v(T \cup \{tx_4\})$. However:

$$\mathrm{gas}^S_{T\cup\{tx_3\}}(tx_3) = \frac{2+2+2+0+0+0}{6} = 1 > \frac{5}{6} = \frac{1+1+1+1+1+0}{6} = \mathrm{gas}^S_{T\cup\{tx_4\}}(tx_4)$$
$$\mathrm{gas}^B_{T\cup\{tx_3\}}(tx_3) = \frac{2+2+0+0}{4} = 1 > \frac{3}{4} = \frac{1+0+1+1}{4} = \mathrm{gas}^B_{T\cup\{tx_4\}}(tx_4)$$
$$\mathrm{gas}^{TPM}_{T\cup\{tx_3\}}(tx_3) = \frac{2}{1+3+2} \cdot 3 = 1 > \frac{4}{5} = \frac{1}{1+3+1} \cdot 4 = \mathrm{gas}^{TPM}_{T\cup\{tx_4\}}(tx_4).$$

So, the Shapley, Banzhaf, and TPM GCMs all violate Scheduling Sensitivity in this case. □

Transaction Bundling (Property 5). We find that the Banzhaf, TPM and XSM GCMs satisfy Transaction Bundling, only XSM satisfying it strictly, while the Shapley and ESM GCMs do not satisfy the property. We prove these facts in the following 5 lemmas.

Lemma 5.3. *The Shapley GCM does* not *satisfy Transaction Bundling (Property 5).*

Lemma 5.4. *The Banzhaf GCM satisfies Transaction Bundling (Property 5).*

Lemma 5.5. *The TPM GCM satisfies Transaction Bundling (Property 5).*

Lemma 5.6. *The ESM GCM does* not *satisfy Transaction Bundling (Property 5).*

Lemma 5.7. *The XSM GCM strictly satisfies Transaction Bundling (Property 5).*

Set Inclusion (Property 4). We find that the TPM and ESM GCMs satisfy Set Inclusion, while the Shapley, Banzhaf and XSM GCMs do not satisfy the property. We prove these facts in the following 5 lemmas.

Lemma 5.8. *The Shapley GCM does* not *satisfy Set Inclusion (Property 4).*

Lemma 5.9. *The Banzhaf GCM does* not *satisfy Set Inclusion (Property 4).*

Lemma 5.10. *The TPM GCM satisfies Set Inclusion (Property 4).*

Lemma 5.11. *The ESM GCM satisfies Set Inclusion (Property 4).*

Lemma 5.12. *The XSM GCM does* not *satisfy Set Inclusion (Property 4).*

Resource-Time Sensitivity (Property 3). All five mechanisms satisfy this property. For the ESM and XSM GCMs, this is an immediate consequence of property (S3). For the Shapley and Banzhaf GCMs, one can see this by recalling that they compute the gas consumption of a transaction $tx \in T$ as a weighted average over $(v(S \cup \{tx\}) - v(S))_{S\subseteq T\setminus\{tx\}}$. Hence, by property (S3), when tx is replaced with some $tx' \gtrsim tx$, no term in the previous decreases, so their weighted average also does not decrease. Finally, for the TPM GCM, we show this in the lemma after the next paragraph.

Resource Sensitivity (Property 1) and Time Sensitivity (Property 2). Because all five mechanisms satisfy Resource-Time Sensitivity, by Lemma 3.1, they also all satisfy Resource Sensitivity and Time Sensitivity. Out of the five mechanisms, the Shapley, Banzhaf, and TPM GCMs satisfy the property strictly. For the first two, this is because when $t(tx)$ increases, at least one term in $(v(S \cup \{tx\}) - v(S))_{S\subseteq T\setminus\{tx\}}$ strictly increases, namely the term for $S = \emptyset$, while no terms decrease by property (S3). Last, for the TPM GCM, we show this in the lemma below.

Lemma 5.13. *The TPM GCM satisfies Resource-Time Sensitivity (Property 3) and strict Time Sensitivity (Property 2).*

6 Towards a Fee Market for Parallel Execution

Armed with an understanding of the achievable trade-offs between desirable properties in a GCM — specifically, the impossibility of satisfying all properties in a single mechanism — and our analysis of various candidate mechanisms and their properties, we propose the *weighted area* GCM for implementation in a fee market for parallel execution. Given that the weighted area GCM satisfies Easy Gas Estimation, it achieves the most additional properties one could hope for in a non-constant GCM (see Theorem 3.4).

Satisfying Easy Gas Estimation ensures that each transaction consumes a fixed amount of gas, regardless of other transactions in the same block. From the perspective of currently deployed TFMs (e.g., EIP-1559), which process transactions with fixed sizes, nothing changes. Thus, the existing properties of these mechanisms remain intact. At a high level, the key properties we strive for in a TFM are incentive compatibility for both block producers and users, welfare optimality, and collusion resistance. However, no TFM can achieve all these properties simultaneously [15, 16, 25]. Importantly, when composing the weighted area GCM with a TFM of choice, the level of sophistication required from users in their bidding strategy does not increase—unlike in currently deployed fee markets for parallel execution [37, 41].

We next further outline some additional steps that need to be taken to implement the weighted area GCM in practice:

Setting Weights. The weighted area GCM uses weights to reflect resource demand, i.e., the higher the demand for a resource, the higher its associated weight. As a result, transactions that are harder to parallelize due to execution on high-demand resources will incur higher gas costs. An open question remains on how to determine the relative weights of these resources. One potential approach is to use an adjustment mechanism similar to those employed in EIP-1559 [10], the blob fee market [11] (an additional Ethereum fee market for data availability), or other proposals aimed at addressing limitations in current demand adjustment mechanisms [5].

Integration with Other Gas Components. Importantly, the weighted area GCM we propose focuses solely on the execution component, whereas gas, in practice, accounts for multiple factors, such as data bandwidth and storage. Moving toward real-world deployment could take two approaches:

— One option is integrating the new execution gas measure into Ethereum's existing framework, which already assigns gas weights to different transaction components (e.g., storage and data bandwidth). These weights are then combined into a single value representing transaction size. To incorporate the weighted area GCM, one would need to determine the relative contribution of execution compared to other gas components and adjust the scaling factors accordingly. With the new weighted area GCM, this task becomes even more challenging, as it is difficult to establish a single value that remains fixed over several years. Specifically, determining the relative cost of reduced parallelization potential from a transaction, compared to the cost of a byte of bandwidth, presents a significant challenge.

— A more principled approach would be to integrate the new execution gas measure into a multi-dimensional fee market, where execution is one component alongside storage, data bandwidth, and other factors. The Ethereum community is actively discussing the transition to a multi-dimensional fee market [8], making this a timely opportunity to incorporate the new execution gas measure. The execution gas component we propose can itself be understood as a multi-dimensional fee market, where each resource represents a distinct dimension with its own posted price, dynamically adjusted based on demand, similarly to the previously outlined approach for weights. These resource dimensions could then be integrated as top-level dimensions within the full multi-dimensional fee market. For example, one dimension might represent bandwidth, while another could correspond to execution on resource r_5. Beyond the general advantages of a multi-dimensional fee market, this approach is perhaps also the most principled and effective method to integrate the *weighted area* GCM into a fee market.

7 Related Work

7.1 Transaction Fee Mechanisms

There is extensive research on blockchain fee markets, with a particular focus on Ethereum and Bitcoin. Early studies primarily examined Bitcoin, exploring monopolistic pricing mechanisms [34, 59]. More recent contributions to this field include [24, 45, 46]. Unlike these works, our study concerns measuring resource usage on a blockchain with client-side parallel execution, rather than focusing on pricing.

The TFM design framework was introduced by Roughgarden [50, 51]. Roughgarden's analysis of the EIP-1559 mechanism [10] initiated an active line of research on TFMs. Chung and Shi [16] demonstrated that no TFM can be ideal — meaning it cannot simultaneously be incentive-compatible for users and block producers while also being resistant to collusion between the two. This conclusion holds even for weaker definitions of collusion resilience, as shown by Chung et al. [15] and Gafni and Yaish [25]. Finally, attempts to address these limitations using cryptographic techniques [55, 58] have made progress in overcoming certain impossibilities, while other attempts relax the desiderata [26]. However, designing an ideal TFM still remains out of reach. While these studies examine the limitations of TFMs, our focus is on GCMs for parallel execution and how to integrate them with a TFM.

A related body of work examines the dynamics of TFMs over multiple blocks, particularly focusing on the base fee in EIP-1559. Leonardos et al. [35, 36] demonstrate that the stability of the base fee depends on the adjustment parameter, with short-term volatility but long-term block size stability. Reijsbergen et al. [49] suggest using an adaptive adjustment parameter to mitigate block size fluctuations, while Ferreira et al. [23] highlight user experience issues caused by bounded base fee oscillations. Additionally, Hougaard and Pourpouneh [30] and Azouvi et al. [5] reveal that the base fee can be manipulated by non-myopic miners.

Given the discussion surrounding multi-dimensional fees in Ethereum [8, 9] and the deployment of EIP-4844 [11] (a first step towards a multi-dimensional fee market on Ethereum), a recent line of work explores multi-dimensional fee markets, focusing on efficient pricing mechanisms and their optimality. This work is further refined by Diamandis et al. [19], who design and analyze multi-dimensional blockchain fee markets to align incentives and improve network performance. Building on this, Angeris et al. [2] prove that such fee markets are nearly optimal, with efficiency improving over time even under adversarial conditions. Multidimensional fee markets are closely related to fee markets designed for parallel execution. In particular, in the weighted area GCM, the weights can be interpreted as fees within a multidimensional fee market. Unlike previous literature on multidimensional fee markets, we focus on parallelization, introduce desirable properties, and evaluate how various mechanisms perform.

Further extensions of TFMs have emerged. Bahrani et al. [7] consider TFMs in the presence of maximal extractable value (MEV), i.e., value extractable by the block producer. Further, Wang et al. [57] design a fee mechanism for proof networks, whereas Bahrani et al. [6] introduce a transaction fee mechanism for heterogeneous computation. Our work most closely relates to the latter, but, in contrast, our chosen approach is closer to multidimensional fee markets, trading complexity for the block producer for stronger incentive compatibility for the user.

Local fee markets have recently been a topic of discussion in the blockchain space [18, 21, 31, 37]. The core idea is that transactions interacting with highly contested states incur higher fees, while those involving non-contested states pay lower fees. However, discussions on local fee markets have largely remained high-level, without a precise characterization of the desired properties beyond this general goal. Moreover, currently implemented local fee markets [37] require significant user sophistication to set fees appropriately. In this work, we formalize the desiderata for fee markets in the context of parallel execution and identify the weighted area GCM as a promising candidate. One key advantage is its compatibility with a TFM, enabling simple fee estimation for users.

7.2 Parallel Execution

Blockchain concurrency has been a focal point in an active line of research. In particular, numerous efforts have aimed to enable parallel transaction processing through speculative execution [1, 3, 14, 20, 27, 52, 54, 60]. Note that speculative execution is already deployed by multiple blockchains [4, 40, 53]. Static analysis has also been employed to identify parallelizable transactions, though it cannot completely eliminate inherent dependencies [42, 47]. Similarly, Neiheiser et al. [44] demonstrate how parallel execution can assist struggling nodes in catching up. While these works are orthogonal to ours, they highlight the overhead of parallel execution when there is no advance knowledge about a transaction's state accesses.

Further, Saraph and Herlihy [52] and Heimbach et al. [28] have evaluated the parallelization potential of the Ethereum workload. The latter demonstrates that a speedup of approximately fivefold is achievable, assuming state accesses are known in advance. Additionally, Solana [56] and Sui [43] already perform parallel execution with advance knowledge of state accesses. However, in practice, state accesses are not known beforehand on many blockchains such as Ethereum. There, less than 2% of transactions disclose them proactively, as shown by Heimbach et al. [29], due to a lack of incentives. In this work, we aim to take a step toward unlocking the parallelization potential by designing a TFM that supports parallel execution. This mechanism relies on the disclosure of state accesses as done in Solana [56] and Sui [43].

8 Conclusion

In this work, we took a step towards creating a fee market that meets the demands of parallel execution environments while also upholding the properties we want from a TFM.

Recently, the idea of local fee markets has been proposed for blockchains that support parallel execution. However, to the best of our knowledge, before this work, the demands on these fee markets have only been outlined at a very high level, and the markets that have been implemented are not ideal yet, e.g., they require high levels of sophistication from users when bidding.

In this work, we addressed this gap by introducing a framework with two key components: a GCM, which measures the execution-related load a transaction imposes on the network in units of gas, and a TFM, which determines the cost associated with each unit of gas. We then formalized the desired properties for the GCM in such a fee market. After outlining the desiderata, we evaluated various mechanisms against them and identified a strong candidate through this analysis — the *weighted area* GCM.

Setting the right incentives in fee markets for parallel execution is crucial to unlocking the full potential of execution layer parallelization, and we hope that our work contributes to the development of fee markets capable of meeting the demands of such environments.

Acknowledgements

We thank the Robust Incentives Group at the Ethereum Foundation for supporting this work through a grant. We thank Boyan Zhou for his work on a student project, supervised by the last three authors, which helped shape this work. Additionally, we are grateful to the attendees of EthereumZüri.ch 2025 for their insightful comments and feedback on this work.

References

[1] Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. 2019. Parblockchain: Leveraging Transaction Parallelism in Permissioned Blockchain Systems. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 1337–1347.

[2] Guillermo Angeris, Theo Diamandis, and Ciamac Moallemi. 2024. Multidimensional Blockchain Fees are (Essentially) Optimal. *arXiv preprint arXiv:2402.08661* (2024).

[3] Parwat Singh Anjana, Sweta Kumari, Sathya Peri, Sachin Rathor, and Archit Somani. 2022. OptSmart: A Space Efficient Optimistic Concurrent Execution of Smart Contracts. *Distributed and Parallel Databases* (2022), 1–53.

[4] Aptos Labs. 2023. Aptos. https://www.aptoslabs.com Accessed: 2025-01-23.

[5] Sarah Azouvi, Guy Goren, Lioba Heimbach, and Alexander Hicks. 2023. Base Fee Manipulation in Ethereum's EIP-1559 Transaction Fee Mechanism. In *37th International Symposium on Distributed Computing*.

[6] Maryam Bahrani and Naveen Durvasula. 2024. Resonance: Transaction Fees for Heterogeneous Computation. *arXiv preprint arXiv:2411.11789* (2024).

[7] Maryam Bahrani, Pranav Garimidi, and Tim Roughgarden. 2024. Transaction Fee Mechanism Design with Active Block Producers. In *International Conference on Financial Cryptography and Data Security*. Springer, 85–90.

[8] Vitalik Buterin. 2022. Multidimensional EIP-1559. https://ethresear.ch/t/multidimensional-eip-1559/11651. Accessed: 2025-01-23.

[9] Vitalik Buterin. 2024. Multidimensional Pricing for EIP-1559. https://vitalik.eth.limo/general/2024/05/09/multidim.html. Accessed: 2025-01-23.

[10] Vitalik Buterin, Eric Conner, Rick Dudley, Matthew Slipper, Ian Norden, and Abdelhamid Bakhta. 2024. EIP-1559: Fee Market Change for ETH 1.0 Chain. https://github.com/ethereum/EIPs/blob/master/EIPS/eip-1559.md. Accessed: 2025-01-23.

[11] Vitalik Buterin, Dankrad Feist, Diederik Loerakker, George Kadianakis, Matt Garnett, Mofi Taiwo, and Ansgar Dietrichs. 2022. EIP-4844: Shard Blob Transactions. https://eips.ethereum.org/EIPS/eip-4844. Accessed: 2025-01-23.

[12] Vitalik Buterin and Martin Swende. 2020. EIP-2930: Optional access lists. https://eips.ethereum.org/EIPS/eip-2930 Accessed: 2025-01-22.

[13] Georgios Chalkiadakis, Edith Elkind, and Michael Wooldridge. 2012. *Weighted Voting Games.* Springer International Publishing, Cham, 49–70. https://doi.org/10.1007/978-3-031-01558-8_5

[14] Yang Chen, Zhongxin Guo, Runhuai Li, Shuo Chen, Lidong Zhou, Yajin Zhou, and Xian Zhang. 2021. Forerunner: Constraint-Based Speculative Transaction Execution for Ethereum. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*. 570–587.

[15] Hao Chung, Tim Roughgarden, and Elaine Shi. 2024. Collusion-Resilience in Transaction Fee Mechanism Design. In *Proceedings of the 25th ACM Conference on Economics and Computation*. 1045–1073.

[16] Hao Chung and Elaine Shi. 2023. Foundations of Transaction Fee Mechanism Design. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 3856–3899.

[17] Xiaotie Deng and Christos H. Papadimitriou. 1994. On the Complexity of Cooperative Solution Concepts. *Mathematics of Operations Research* 19, 2 (1994), 257–266. https://doi.org/10.1287/moor.19.2.257

[18] Theo Diamandis, Tarun Chitra, and 0xShitTrader. 2024. Toward Multidimensional Solana Fees. *Umbra Research* (2024). https://www.umbraresearch.xyz/writings/toward-multidimensional-solana-fees

[19] Theo Diamandis, Alex Evans, Tarun Chitra, and Guillermo Angeris. 2023. Designing Multidimensional Blockchain Fee Markets. In *5th Conference on Advances in Financial Technologies (AFT 2023)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[20] Thomas Dickerson, Paul Gazzillo, Maurice Herlihy, and Eric Koskinen. 2017. Adding Concurrency to Smart Contracts. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*. 303–312.

[21] Eclipse Labs. 2024. Local Fee Markets are Necessary to Scale Ethereum. *Eclipse* (2024). https://www.eclipse.xyz/articles/local-fee-markets-are-necessary-to-scale-ethereum

[22] Piotr Faliszewski and Lane Hemaspaandra. 2009. The complexity of power-index comparison. *Theoretical Computer Science* 410, 1 (2009), 101–107. https://doi.org/10.1016/j.tcs.2008.09.034

[23] Matheus VX Ferreira, Daniel J Moroz, David C Parkes, and Mitchell Stern. 2021. Dynamic Posted-Price mMchanisms for

the Blockchain Transaction-Fee Market. In *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*. 86–99.

[24] Yotam Gafni and Aviv Yaish. 2022. Greedy Transaction Fee Mechanisms for (Non-) Myopic Miners. *arXiv preprint arXiv:2210.07793* 5 (2022).

[25] Yotam Gafni and Aviv Yaish. 2024. Barriers to Collusion-Resistant Transaction Fee Mechanisms. In *Proceedings of the 25th ACM Conference on Economics and Computation*. 1074–1096.

[26] Yotam Gafni and Aviv Yaish. 2024. Discrete and Bayesian Transaction Fee Mechanisms. In *The International Conference on Mathematical Research for Blockchain Economy*. Springer, 145–171.

[27] Rati Gelashvili, Alexander Spiegelman, Zhuolun Xiang, George Danezis, Zekun Li, Yu Xia, Runtian Zhou, and Dahlia Malkhi. 2022. Block-STM: Scaling Blockchain Execution by Turning Ordering Curse to a Performance Blessing. *arXiv preprint arXiv:2203.06871* (2022).

[28] Lioba Heimbach, Quentin Kniep, Yann Vonlanthen, and Roger Wattenhofer. 2023. DeFi and NFTs Hinder Blockchain Scalability. In *International Conference on Financial Cryptography and Data Security*. Springer, 291–309.

[29] Lioba Heimbach, Quentin Kniep, Yann Vonlanthen, Roger Wattenhofer, and Patrick Züst. 2023. Dissecting the EIP-2930 Optional Access Lists. *arXiv preprint arXiv:2312.06574* (2023).

[30] Jens Leth Hougaard and Mohsen Pourpouneh. 2023. Farsighted Miners Under Transaction Fee Mechanism EIP-1559. In *2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 1–9.

[31] keyneom. 2024. Local Fee Markets in Ethereum. *Ethereum Research* (2024). https://ethresear.ch/t/local-fee-markets-in-ethereum/20754

[32] Mysten Labs. 2025. Sui Object Model Documentation. https://docs.sui.io/concepts/object-model Accessed: 2025-01-22.

[33] Solana Labs. 2025. Solana Transactions Documentation. https://solana.com/de/docs/core/transactions Accessed: 2025-01-22.

[34] Ron Lavi, Or Sattath, and Aviv Zohar. 2022. Redesigning Bitcoin's Fee Market. *ACM Transactions on Economics and Computation* 10, 1 (2022), 1–31.

[35] Stefanos Leonardos, Barnabé Monnot, Daniël Reijsbergen, Efstratios Skoulakis, and Georgios Piliouras. 2021. Dynamical Analysis of the EIP-1559 Ethereum Fee Market. In *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*. 114–126.

[36] Stefanos Leonardos, Daniël Reijsbergen, Barnabé Monnot, and Georgios Piliouras. 2023. Optimality Despite Chaos in Fee Markets. In *International Conference on Financial Cryptography and Data Security*. Springer, 346–362.

[37] Lostin. 2025. The Truth about Solana Local Fee Markets. *Helius* (2025). https://www.helius.dev/blog/solana-local-fee-markets

[38] Tomomi Matsui and Yasuko Matsui. 2000. A Survey of Algorithms for Calculating Power Indices of Weighted Majority Games. *Journal of the Operations Research Society of Japan* 43 (Nov 2000), 71–86. https://doi.org/10.15807/jorsj.43.71

[39] Yasuko Matsui and Tomomi Matsui. 2001. NP-completeness for calculating power indices of weighted majority games. *Theoretical Computer Science* 263, 1 (2001), 305–310. https://doi.org/10.1016/S0304-3975(00)00251-6 Combinatorics and Computer Science.

[40] Monad Labs. 2023. Monad. https://www.monad.xyz Accessed: 2025-01-23.

[41] Sebastian Mueller. 2025. X post by Sebastian Mueller. https://x.com/NaitsabesMue/status/1862519048069959893. Accessed: 2025-02-08.

[42] Maurizio Murgia, Letterio Galletta, and Massimo Bartoletti. 2021. A Theory of Transaction Parallelism in Blockchains. *Logical Methods in Computer Science* 17 (2021).

[43] Mysten Labs. 2023. Sui. https://sui.io Accessed: 2025-01-23.

[44] Ray Neiheiser, Arman Babaei, Ioannis Alexopoulos, Marios Kogias, and Eleftherios Kokoris Kogias. 2024. Pythia: Supercharging Parallel Smart Contract Execution to Guide Stragglers and Full Nodes to Safety. https://aftsib.com/papers/SIB24_paper_4.pdf. Accessed: 2025-01-23.

[45] Noam Nisan. 2023. Serial Monopoly on Blockchains. *arXiv preprint arXiv:2311.12731* (2023).

[46] Paolo Penna and Manvir Schneider. 2024. Serial Monopoly on Blockchains with Quasi-patient Users. *arXiv preprint arXiv:2405.17334* (2024).

[47] George Pîrlea, Amrit Kumar, and Ilya Sergey. 2021. Practical Smart Contract Sharding with Ownership and Commutativity Analysis. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. 1327–1341.

[48] K. Prasad and J. S. Kelly. 1990. NP-completeness of some problems concerning voting games. *International Journal of Game Theory* 19, 1 (Mar 1990), 1–9. https://doi.org/10.1007/BF01753703

[49] Daniël Reijsbergen, Shyam Sridhar, Barnabé Monnot, Stefanos Leonardos, Stratis Skoulakis, and Georgios Piliouras. 2021. Transaction Fees on a Honeymoon: Ethereum's EIP-1559 One Month Later. In *2021 IEEE International Conference on Blockchain (Blockchain)*. IEEE, 196–204.

[50] Tim Roughgarden. 2020. Transaction Fee Mechanism Design for the Ethereum Blockchain: An Economic Analysis of EIP-1559. *arXiv preprint arXiv:2012.00854* (2020).
[51] Tim Roughgarden. 2024. Transaction Fee Mechanism Design. *J. ACM* 71, 4 (2024), 1–25.
[52] Vikram Saraph and Maurice Herlihy. 2019. An Empirical Study of Speculative Concurrency in Ethereum Smart Contracts. *arXiv preprint arXiv:1901.01376* (2019).
[53] Sei Labs. 2023. Sei Protocol. https://www.sei.io Accessed: 2025-01-23.
[54] Ilya Sergey and Aquinas Hobor. 2017. A Concurrent Perspective on Smart Contracts. In *International Conference on Financial Cryptography and Data Security*. Springer, 478–493.
[55] Elaine Shi, Hao Chung, and Ke Wu. 2022. What can Cryptography do for Decentralized Mechanism Design. *arXiv preprint arXiv:2209.14462* (2022).
[56] Solana Labs. 2023. Solana. https://solana.com Accessed: 2025-01-23.
[57] Wenhao Wang, Lulu Zhou, Aviv Yaish, Fan Zhang, Ben Fisch, and Benjamin Livshits. 2024. Mechanism Design for ZK-Rollup Prover Markets. *arXiv preprint arXiv:2404.06495* (2024).
[58] Ke Wu, Elaine Shi, and Hao Chung. 2023. Maximizing Miner Revenue in Transaction Fee Mechanism Design. *arXiv preprint arXiv:2302.12895* (2023).
[59] Andrew Chi-Chih Yao. 2020. An Incentive Analysis of some Bitcoin Fee Designs. In *Proceedings of the 47th International Colloquium on Automata, Languages, and Programming (ICALP)*.
[60] An Zhang and Kunlong Zhang. 2018. 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*. Springer, 425–439.

A Proofs Omitted From Section 3

Lemma 3.1. *Property 3 holds if and only if Properties 1 and 2 hold.*

Proof. Properties 1 and 2 are special cases of Property 3, corresponding to the scenarios where $t_1 = t_2$ and $R_1 = R_2$, respectively. Therefore, the ($\Rightarrow$) direction holds. To prove the ($\Leftarrow$) direction, assume Properties 1 and 2 hold and consider a set of transactions T and two transactions $tx_1 \simeq (t_1, R_1)$ and $tx_2 \simeq (t_2, R_2)$, both not in T, such that $t_1 \leq t_2$ and $R_1 \subseteq R_2$. Let tx_3 be another transaction not in T such that $tx_3 \simeq (t_1, R_2)$. By first applying Property 1 and then Property 2, we obtain:

$$\text{gas}_{T\cup\{tx_1\}}(tx_1) \leq \text{gas}_{T\cup\{tx_3\}}(tx_3) \leq \text{gas}_{T\cup\{tx_2\}}(tx_2).$$

This establishes the conclusion. □

Lemma 3.2. *Property 3 holds strictly if and only if Properties 1 and 2 hold strictly.*

Proof. The strict variants of Properties 1 and 2 are special cases of the strict variant of Property 3, corresponding to the scenarios where $t_1 = t_2$ and $R_1 = R_2$, respectively. Therefore, the ($\Rightarrow$) direction holds. To prove the ($\Leftarrow$) direction, assume Properties 1 and 2 hold strictly and consider a set of transactions T and two transactions $tx_1 \simeq (t_1, R_1)$ and $tx_2 \simeq (t_2, R_2)$, both not in T, such that $t_1 \leq t_2$ and $R_1 \subseteq R_2$, and additionally $tx_1 \not\simeq tx_2$ (i.e., at least one of the previous holds strictly). Let tx_3 be another transaction not in T such that $tx_3 \simeq (t_1, R_2)$. By first applying Property 1 and then Property 2 (their *non-strict* versions), we obtain:

$$\text{gas}_{T\cup\{tx_1\}}(tx_1) \leq \text{gas}_{T\cup\{tx_3\}}(tx_3) \leq \text{gas}_{T\cup\{tx_2\}}(tx_2). \tag{5}$$

From this, we get that $\text{gas}_{T\cup tx_1}(tx_1) \leq \text{gas}_{T\cup tx_2}(tx_2)$, so it remains to rule out the equality case. Assume for a contradiction that $\text{gas}_{T\cup\{tx_1\}}(tx_1) = \text{gas}_{T\cup\{tx_2\}}(tx_2)$, from which the two inequalities hold with equality in Eq. (5). Because Properties 1 and 2 hold strictly, this can only be the case if $tx_1 \simeq tx_3 \simeq tx_2$, contradicting the assumption that $tx_1 \not\simeq tx_2$. □

B Proofs Omitted From Section 4

Lemma 4.5. *The Banzhaf GCM does* not *satisfy Efficiency (Property 7).*

Proof. Consider the set of transactions $T = \{tx_1, tx_2, tx_3\}$ where $tx_1 \simeq (1, \{r_1\})$, $tx_2 \simeq (1, \{r_1\})$, and $tx_3 \simeq (1, \{r_2\})$. Then, for any number of threads $n \geq 2$ and the optimal scheduler, we have:

$$v(\{tx_1\}) = v(\{tx_2\}) = v(\{tx_3\}) = v(\{tx_1, tx_3\}) = v(\{tx_2, tx_3\}) = 1, \quad v(\{tx_1, tx_2\}) = v(T) = 2.$$

A short calculation then shows that:

$$\text{gas}^B_T(tx_1) = \text{gas}^B_T(tx_2) = \frac{1+1+0+1}{4} = \frac{3}{4}, \quad \text{gas}^B_T(tx_3) = \frac{1+0+0+0}{4} = \frac{1}{4}.$$

Thus, $\sum_{tx\in T} \text{gas}^B_T(tx) = \frac{3}{4} + \frac{3}{4} + \frac{1}{4} = \frac{7}{4} \neq 2 = v(T)$, violating Efficiency. □

C Proofs Omitted From Section 5

C.1 Transaction Bundling (Property 5)

Lemma 5.3. *The Shapley GCM does* not *satisfy Transaction Bundling (Property 5).*

Proof. Consider the set of transactions $T = \{tx_4\}$ and three other transactions tx_1, tx_2, tx_3 such that tx_3 is the concatenation of tx_1 and tx_2, where $tx_1 \simeq (1, \{r_2\})$, $tx_2 \simeq (1, \{r_2\})$, $tx_3 \simeq (2, \{r_2\})$, and $tx_4 \simeq (1, \{r_1\})$. Then, for any number of threads $n \geq 2$ and the optimal scheduler, we have:

$$\text{gas}^S_{T\cup\{tx_1,tx_2\}}(tx_1) = \text{gas}^S_{T\cup\{tx_1,tx_2\}}(tx_2) = \frac{0+1+1+1+1+1}{6}$$

$$\text{gas}^S_{T\cup\{tx_3\}}(tx_3) = \frac{2+1}{2}.$$

Thus, $\text{gas}^S_{T\cup\{tx_1,tx_2\}}(tx_1) + \text{gas}^S_{T\cup\{tx_1,tx_2\}}(tx_2) = \frac{10}{6} > \frac{3}{2} = \text{gas}^S_{T\cup\{tx_3\}}(tx_3)$, which violates Transaction Bundling. $\square$

Lemma 5.4. *The Banzhaf GCM satisfies Transaction Bundling (Property 5).*

Proof. Consider a set of transactions T and three transactions $tx_1, tx_2, tx_3 \notin T$ such that tx_3 is the concatenation of tx_1 and tx_2. Then, for any scheduler satisfying property (S2), we have:

$$\begin{aligned}
&\text{gas}^B_{T\cup\{tx_1,tx_2\}}(tx_1) + \text{gas}^B_{T\cup\{tx_1,tx_2\}}(tx_2)\\
&\quad= \frac{1}{2^{|T\cup\{tx_1,tx_2\}|-1}} \sum_{S\subseteq (T\cup\{tx_1,tx_2\})\setminus\{tx_1\}} [v(S\cup\{tx_1\}) - v(S)]\\
&\quad+ \frac{1}{2^{|T\cup\{tx_1,tx_2\}|-1}} \sum_{S\subseteq (T\cup\{tx_1,tx_2\})\setminus\{tx_2\}} [v(S\cup\{tx_2\}) - v(S)]\\
&\quad= \frac{1}{2^{|T\cup\{tx_1,tx_2\}|-1}} \sum_{S\subseteq T} [v(S\cup\{tx_1\}) - v(S) + v(S\cup\{tx_1,tx_2\}) - v(S\cup\{tx_2\})]\\
&\quad+ \frac{1}{2^{|T\cup\{tx_1,tx_2\}|-1}} \sum_{S\subseteq T} [v(S\cup\{tx_2\}) - v(S) + v(S\cup\{tx_1,tx_2\}) - v(S\cup\{tx_1\})]\\
&\quad= \frac{2}{2^{|T\cup\{tx_1,tx_2\}|-1}} \sum_{S\subseteq T} [v(S\cup\{tx_1,tx_2\}) - v(S)]\\
&\quad\leq \frac{1}{2^{|T\cup\{tx_3\}|-1}} \sum_{S\subseteq T} [v(S\cup\{tx_3\}) - v(S)] \quad = \quad \text{gas}^B_{T\cup\{tx_3\}}(tx_3).
\end{aligned}$$

$\square$

Lemma 5.5. *The TPM GCM satisfies Transaction Bundling (Property 5).*

Proof. Consider a set of transactions T and three transactions $tx_1, tx_2, tx_3 \notin T$ such that tx_3 is the concatenation of tx_1 and tx_2. Then, for any scheduler satisfying property (S2), we have:

$$\begin{aligned}
&\text{gas}^{TPM}_{T\cup\{tx_1,tx_2\}}(tx_1) + \text{gas}^{TPM}_{T\cup\{tx_1,tx_2\}}(tx_2)\\
&\quad= \frac{t(tx_1)}{\sum_{tx'\in T\cup\{tx_1,tx_2\}} t(tx')} \cdot v(T\cup\{tx_1,tx_2\}) + \frac{t(tx_2)}{\sum_{tx'\in T\cup\{tx_1,tx_2\}} t(tx')} \cdot v(T\cup\{tx_1,tx_2\})\\
&\quad= \frac{t(tx_1)+t(tx_2)}{\sum_{tx'\in T\cup\{tx_1,tx_2\}} t(tx')} \cdot v(T\cup\{tx_1,tx_2\})\\
&\quad= \frac{t(tx_3)}{\sum_{tx'\in T\cup\{tx_3\}} t(tx')} \cdot v(T\cup\{tx_1,tx_2\})\\
&\quad\leq \frac{t(tx_3)}{\sum_{tx'\in T\cup\{tx_3\}} t(tx')} \cdot v(T\cup\{tx_3\}) \quad = \quad \text{gas}^{TPM}_{T\cup\{tx_3\}}(tx_3).
\end{aligned}$$

$\square$

Lemma 5.6. *The ESM GCM does* not *satisfy Transaction Bundling (Property 5).*

Proof. Consider the set of transactions $T = \{tx_4\}$ and three transactions $tx_1, tx_2, tx_3 \notin T$ such that tx_3 is the concatenation of tx_1 and tx_2, where $tx_1 \simeq (1, \{r_1\})$, $tx_2 \simeq (1, \{r_1\})$, $tx_3 \simeq (2, \{r_1\})$, and $tx_4 \simeq (1, \{r_1\})$. Then, for any number of threads $n \geq 2$ and the optimal scheduler, we have:

$$\text{gas}^{ESM}_{T\cup\{tx_1,tx_2\}}(tx_1) = \text{gas}^{ESM}_{T\cup\{tx_1,tx_2\}}(tx_2) = \frac{3}{3}, \quad \text{gas}^{ESM}_{T\cup\{tx_3\}}(tx_3) = \frac{3}{2}.$$

Thus, $\text{gas}^{ESM}_{T\cup\{tx_1,tx_2\}}(tx_1) + \text{gas}^{ESM}_{T\cup\{tx_1,tx_2\}}(tx_2) = 2 > \frac{3}{2} = \text{gas}^{ESM}_{T\cup\{tx_3\}}(tx_3)$, which violates Transaction Bundling. $\square$

Lemma 5.7. *The XSM GCM strictly satisfies Transaction Bundling (Property 5).*

Proof. Consider a set of transactions T and three transactions $tx_1, tx_2, tx_3 \notin T$ such that tx_3 is the concatenation of tx_1 and tx_2. Then, for any scheduler satisfying property (S2), we have:

$$\begin{aligned}
&\text{gas}^{XSM}_{T\cup\{tx_1,tx_2\}}(tx_1) + \text{gas}^{XSM}_{T\cup\{tx_1,tx_2\}}(tx_2) \\
&\qquad = \frac{v(T\cup\{tx_1,tx_2\})}{3^{|T\cup\{tx_1,tx_2\}|}} + \frac{v(T\cup\{tx_1,tx_2\})}{3^{|T\cup\{tx_1,tx_2\}|}} \\
&\qquad = \frac{2}{3}\cdot\frac{v(T\cup\{tx_1,tx_2\})}{3^{|T\cup\{tx_3\}|}} \\
&\qquad < \frac{v(T\cup\{tx_1,tx_2\})}{3^{|T\cup\{tx_3\}|}} \\
&\qquad \le \frac{v(T\cup\{tx_3\})}{3^{|T\cup\{tx_3\}|}} = \text{gas}^{XSM}_{T\cup\{tx_3\}}(tx_3).
\end{aligned}$$

□

C.2 Set Inclusion (Property 4)

Lemma 5.8. *The Shapley GCM does* not *satisfy Set Inclusion (Property 4).*

Proof. Consider the transaction sets $T = \{tx_1\}$, $T_1 = \{tx_2, tx_3\}$, and $T_2 = \{tx_2, tx_3, tx_4\}$ where $tx_1 \simeq (1, \{r_1\})$, $tx_2 \simeq (1, \{r_2\})$, $tx_3 \simeq (1, \{r_2\})$, and $tx_4 \simeq (1, \{r_1\})$. Then, for any number of threads $n \ge 2$ and the optimal scheduler, we have:

$$\text{gas}^{S}_{T\cup T_1}(T_1) = \frac{5}{6} + \frac{5}{6}, \qquad \text{gas}^{S}_{T\cup T_2}(T_2) = \frac{12}{24} + \frac{12}{24} + \frac{12}{24}.$$

Thus, $\text{gas}^{S}_{T\cup T_1}(T_1) = \frac{10}{6} > \frac{36}{24} = \text{gas}^{S}_{T\cup T_2}(T_2)$, which violates Set Inclusion. □

Lemma 5.9. *The Banzhaf GCM does* not *satisfy Set Inclusion (Property 4).*

Proof. Consider the transaction sets $T = \emptyset$, $T_1 = \{tx_1, tx_2\}$, and $T_2 = \{tx_1, tx_2, tx_3\}$ where $tx_1 \simeq (1, \{r_1\})$, $tx_2 \simeq (1, \{r_1\})$, and $tx_3 \simeq (1, \{r_2\})$. Then, for any number of threads $n \ge 2$ and the optimal scheduler, we have:

$$\text{gas}^{B}_{T\cup T_1}(T_1) = \frac{1+1}{2} + \frac{1+1}{2}, \qquad \text{gas}^{B}_{T\cup T_2}(T_2) = \frac{1+1+0+1}{4} + \frac{1+1+0+1}{4} + \frac{1+0+0+0}{4}.$$

Thus, $\text{gas}^{B}_{T\cup T_1}(T_1) = 2 > \frac{7}{4} = \text{gas}^{B}_{T\cup T_2}(T_2)$, which violates Set Inclusion. □

Lemma 5.10. *The TPM GCM satisfies Set Inclusion (Property 4).*

Proof. Consider a set of transactions T and two sets of transactions $T_1 \subseteq T_2$, disjoint from T. Then, for any scheduler satisfying property (S1), we have:

$$\begin{aligned}
\text{gas}^{TPM}_{T\cup T_1}(T_1) &= \frac{\sum_{tx\in T_1} t(tx)}{\sum_{tx'\in T\cup T_1} t(tx')}\cdot v(T\cup T_1) \\
&\le \frac{\sum_{tx\in T_1} t(tx) + \sum_{tx\in T_2\setminus T_1} t(tx)}{\sum_{tx'\in T\cup T_1} t(tx') + \sum_{tx'\in T_2\setminus T_1} t(tx')}\cdot v(T\cup T_1) \\
&= \frac{\sum_{tx\in T_2} t(tx)}{\sum_{tx'\in T\cup T_2} t(tx')}\cdot v(T\cup T_1) \\
&\le \frac{\sum_{tx\in T_2} t(tx)}{\sum_{tx'\in T\cup T_2} t(tx')}\cdot v(T\cup T_2) \quad = \quad \text{gas}^{TPM}_{T\cup T_2}(T_2).
\end{aligned}$$

□

Lemma 5.11. *The ESM GCM satisfies Set Inclusion (Property 4).*

Proof. Consider a set of transactions T and two sets of transactions $T_1 \subseteq T_2$, disjoint from T. Then, for any scheduler satisfying property (S1), we have:

$$\begin{aligned}
\text{gas}^{ESM}_{T\cup T_1}(T_1) &= \sum_{tx\in T_1} \frac{v(T\cup T_1)}{|T\cup T_1|} \\
&= \frac{|T_1|}{|T\cup T_1|}\cdot v(T\cup T_1) \\
&\leq \frac{|T_1| + |T_2\backslash T_1|}{|T\cup T_1| + |T_2\backslash T_1|}\cdot v(T\cup T_1) \\
&= \frac{|T_2|}{|T\cup T_2|}\cdot v(T\cup T_1) \\
&\leq \frac{|T_2|}{|T\cup T_2|}\cdot v(T\cup T_2) \quad = \quad \text{gas}^{ESM}_{T\cup T_2}(T_2).
\end{aligned}$$

□

Lemma 5.12. *The XSM GCM does* not *satisfy Set Inclusion (Property 4).*

Proof. Consider the transaction sets $T = \emptyset$, $T_1 = \{tx_1\}$, and $T_2 = \{tx_1, tx_2\}$, where $tx_1 \simeq (1, \{r_1\})$ and $tx_2 \simeq (1, \{r_2\})$. Then, for any number of threads $n \geq 2$ and the optimal scheduler, we have:

$$\text{gas}^{XSM}_{T\cup T_1}(T_1) = \frac{1}{3^1}, \qquad \text{gas}^{XSM}_{T\cup T_2}(T_2) = \frac{1}{3^2} + \frac{1}{3^2}.$$

Thus, $\text{gas}^{XSM}_{T\cup T_1}(T_1) = \frac{1}{3} > \frac{2}{9} = \text{gas}^{XSM}_{T\cup T_2}(T_2)$, which violates Set Inclusion. □

C.3 Resource-Time Sensitivity (Property 3) and Strict Time Sensitivity (Property 2)

Lemma 5.13. *The TPM GCM satisfies Resource-Time Sensitivity (Property 3) and strict Time Sensitivity (Property 2).*

Proof. Consider a set of transactions T and two transactions $tx_1 \simeq (t_1, R_1)$ and $tx_2 \simeq (t_2, R_2)$, both not in T, such that $t_1 \leq t_2$ and $R_1 \subseteq R_2$. Then, for any scheduler satisfying property (S3):

$$\begin{aligned}
\text{gas}^{TPM}_{T\cup\{tx_1\}}(tx_1) &= \frac{t_1}{\sum_{tx'\in T\cup\{tx_1\}} t(tx')}\cdot v(T\cup\{tx_1\}) \\
&= \frac{t_1}{\sum_{tx'\in T} t(tx') + t_1}\cdot v(T\cup\{tx_1\}) \\
&\leq \frac{t_2}{\sum_{tx'\in T} t(tx') + t_2}\cdot v(T\cup\{tx_1\}) \\
&\leq \frac{t_2}{\sum_{tx'\in T} t(tx') + t_2}\cdot v(T\cup\{tx_2\}) \quad = \quad \text{gas}^{TPM}_{T\cup\{tx_2\}}(tx_2).
\end{aligned}$$

This establishes Resource-Time Sensitivity.

To also get strict Time Sensitivity, assume $t_1 < t_2$ and $R_1 = R_2$ in the above proof. Note that, when $\sum_{tx'\in T} t(tx') > 0$, the function $\frac{t}{\sum_{tx'\in T} t(tx')+t}$ is strictly increasing in t, guaranteeing that the first inequality above is strict, so $\text{gas}^{TPM}_{T\cup\{tx_1\}}(tx_1) < \text{gas}^{TPM}_{T\cup\{tx_2\}}(tx_2)$, as required. In the degenerate case where $\sum_{tx'\in T} t(tx') = 0$, the fraction becomes 1 in both cases, but this can only happen when $T = \emptyset$, in which case the value v increases strictly because $v(\{tx_1\}) = t_1 < t_2 = v(\{tx_2\})$. Overall, the first inequality above again holds strictly, ensuring that $\text{gas}^{TPM}_{T\cup\{tx_1\}}(tx_1) < \text{gas}^{TPM}_{T\cup\{tx_2\}}(tx_2)$ still holds, as required. Thus, the strict Time Sensitivity property is satisfied. □