POSTER: OCToPus: Semantic-aware Concurrency Control for Blockchain Transactions

dePaul Miller
Lehigh University
dsm220@lehigh.edu

Hank Korth
Lehigh University
hfk2@lehigh.edu

Roberto Palmeiri
Lehigh University
palmieri@lehigh.edu

Abstract

Many blockchain implementations offer APIs to send and receive money between accounts exclusively. In this paper, we introduce OCToPus, a deterministic concurrency control scheme that uses a semantic-aware fast path and a GPU-accelerated directed acyclic graph-based fallback path to parallelize the execution of a block aggressively.

CCS Concepts: • **Information systems → Database transaction processing**.

Keywords: Blockchain, Concurrency Control, GPU

ACM Reference Format:
dePaul Miller, Hank Korth, and Roberto Palmeiri. 2024. POSTER: OCToPus: Semantic-aware Concurrency Control for Blockchain Transactions. In *The 29th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming (PPoPP '24), March 2–6, 2024, Edinburgh, United Kingdom.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3627535.3638494

1 Introduction

Blockchains are becoming an increasingly prevalent type of secure distributed transactional processing scheme for large-scale systems. Unquestionably, its popularity comes from its ability to implement payment systems [12, 13, 21] and possibly enable the establishment of central bank digital currency [1, 11]. In order to execute distributed transactions in a typical blockchain *1)*, a proposer pulls transactions from the mempool (i.e., a transaction repository) in some order to form and validates a block before proposing it; *2)*, a consensus algorithm is then responsible for agreeing on that block; *3)*, and validators receive that block and execute it deterministically to validate it and replicate the blockchain state, ensuring immutability and irrefutability. In this paper, we introduce a transaction processing scheme that can be used by proposers before submitting the block to consensus and by validators during consensus to maximize the number of transactions that are committed in a block.

PPoPP '24, March 2–6, 2024, Edinburgh, United Kingdom

ACM ISBN 979-8-4007-0435-2/24/03.
https://doi.org/10.1145/3627535.3638494

We want to focus on two important characteristics of the above process. First, the outcome of processing transactions in a block must be deterministic so each node of the blockchain can reproduce the shared state after executing a sequence of blocks. Second, many blockchain implementations, especially those permissionless, focus on financial transactions (i.e., moving money between accounts) [3, 4, 8, 19] and have a rigid programming model.

The high performance of the concurrency control that ensures the deterministic transaction processing schema is essential for implementing competitive blockchains. Prior work [9, 14, 17] has considered various ways to parallelize this execution. These techniques must detect and resolve conflicts (e.g., anti-dependency [6] or blind write [10]) to process transactions in a way that is equivalent to running them sequentially in the pre-defined order. The overhead entailed by these advanced synchronization schemes often nullifies the benefits of the parallel computation [16, 18, 20].

We introduce Optimistic Commutative and compensating Transaction Processing (OCToPus) to *a)* leverage the strict programming model of blockchain implementations and the absence of interactive transactions, and *b)* effectively exploit the commutativity property of transactional operations to improve performance.

2 OCToPus

We consider a blockchain model where transactions operate on accounts, and the blockchain maintains account balances rather than tracking unspent transaction output, as done by Bitcoin [13]. Our assumed model follows that of Ethereum [21] and other chains [2, 3], including blockchain-based decentralized applications such as Uniswap [5].

The set of transactions utilized in the blockchains adopting our model comes from a service called a mempool, which is created through a gossip protocol. Proposers in the blockchain may pick any set of transactions and propose them through a consensus protocol. The choice of consensus protocol is orthogonal to OCToPus. Unlike more traditional concurrency control protocols (e.g., 2PL [7]), we require determinism in order to replicate the shared state correctly.

OCToPus is designed for blockchain systems. Most of these systems [3, 4, 8] have a rigid programming model where money is exchanged between accounts through simple transactions that involve increasing and decreasing values (e.g., balance and/or some metadata such as a sequence number [8]) associated with various accounts. For simplicity,

we name this model *send-receive-money* (or SRM in short). Because of the nature of SRM, it is possible to define compensating actions that revert the effect of an unwanted commit.

We utilize the semantics of SRM to break down each transaction into a sequence of atomic operations and run the atomic operations of different transactions in parallel. We call this the *fast path*. When we recognize a transaction's execution is no longer equivalent to its original semantics (e.g., funds were not available to perform a transfer in the first place), we undo it by executing compensating actions and re-execute it by relying on a graph-based concurrency control, optimized for GPU. We call this the *fallback path.*

2.1 Fast Path

In the SRM programming model, transactions are committed (money is sent) only if the sender has a great enough balance. This model allows for many transactions to commute (i.e., the state of the data repository after running commutative transactions is equivalent regardless of the ordering of said transactions). In the SRM programming model, if we allow conflicting yet commutative transactions to interleave, it is enough to execute each individual operation atomically to produce a schedule where no account ends with insufficient funds to perform all the transfers in the block.

However, even in a blockchain, transactions do not always commute. In this case, the order of transactions will determine which transaction succeeds and which aborts. It is important to note that unavailability of funds is the *only* reason for a transaction to abort in OCToPus. Because of this, our concurrency control in the fast path does not need to keep track of accesses performed by transactions. When an abort occurs in the fast path, OCToPus must ensure that its modification to the sender's account is undone before re-execution. That is because transactions are broken down into multiple atomic operations, and therefore, their effects are immediately applied to the shared state.

In the blockchain, many client transactions are batched into a block, no transactions are interactively submitted, and no output is externalized to clients unless the entire block is committed. That means, even if a transaction applies its modifications to the shared state and then aborts, no other transaction but those in the block can access the change. As a result, by tracking transactions' dependencies, undoing the effect of a transaction that aborts due to the unavailability of funds using cascading rollback prevents final inconsistent states. The fallback path ensures this.

2.2 Fallback Path

The fallback path is designed for handling non-commutative transactions. We consider the ordering in the block as a reference order for conflicting transactions and build a directed acyclic graph (DAG) to order them accordingly. Recall that there is a conflict between two transactions if both access the same data, and at least one of them writes it concurrently. These conflicts are represented as directed edges where a transaction T_1 that occurs before in the block order that conflicts with a transaction T_2 that occurs later in the block order will have an edge from T_1 to T_2 in the DAG. Because of the simplicity of the SRM model, accounts' IDs are provided by the programmer and not defined at runtime. With this assumption, we can identify a-priori transactions' access sets (i.e., their read-set and write-set) by parsing the transaction APIs for reading and writing accounts.

We move from the fast path to the fallback path if there are aborts. The set of aborted transactions is given to the fallback path. Compensating transactions enable us to undo the modifications performed by these aborted transactions efficiently. One transaction, T_2, compensates another transaction, T_1, if running the transaction T_1 and then T_2 results in the same state prior to the execution of T_1. In the SRM model, compensating for an aborted transaction means depositing money, which is an operation that always succeeds.

Not only the transactions aborted in the fast path are undone, but also all their dependent transactions, as identified in the DAG. Compensated transactions are then processed in the fallback path, following the order imposed by the DAG. Non-conflicting sets of transactions in the DAG are not connected, and each subgraph that is not connected can be executed in parallel. Transactions in a subgraph are executed sequentially to avoid conflict resolution, but since they can still abort due to the unavailability of funds, their writes are buffered until the transaction's completion.

Although the DAG-based schema is well known in literature [15, 19], the novelty of our approach lies in using it as a fallback for non-commutative transactions and redesigning it to take advantage of the GPU parallelism. OCToPus is able to utilize the GPU to build an upper triangular matrix that represents this DAG efficiently. This process is accomplished in parallel while the CPU executes the fast path.

3 Conclusion

We briefly introduced OCToPus, a deterministic transaction processing scheme that speeds up the execution of blocks of transactions in blockchain implementations that offer API to transfer money between accounts exclusively. OCToPus can be used by blockchain proposer and validator nodes when dealing with transactions that transfer money between accounts. OCToPus aggressively exploits the commutativity property of these simple transactions to eliminate the need for instrumentation at runtime.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. CNS-2045976. This research was also funded by a CORE grant from Lehigh University and by a gift grant from the Stellar Dev. Foundation.

References

[1] 2022. Federal Reserve Board releases discussion paper that examines pros and cons of a potential U.S. central bank digital currency (CBDC). https://www.federalreserve.gov/newsevents/pressreleases/other20220120a.htm

[2] 2023. Aptos Developer Documentation. https://aptos.dev/

[3] 2023. Operations and Transactions. https://developers.stellar.org/docs/fundamentals-and-concepts/stellar-data-structures/operations-and-transactions

[4] 2023. Transactions. https://developer.algorand.org/docs/get-details/transactions

[5] Hayden Adams, Noah Zinsmeister, Moody Salem, River Keefer, and Dan Robinson. 2021. Uniswap v3 Core. https://uniswap.org/whitepaper-v3.pdf

[6] Atul Adya. 1999. Weak consistency: a generalized theory and optimistic implementations for distributed transactions. (1999).

[7] Kapali P. Eswaran, Jim Gray, Raymond A. Lorie, and Irving L. Traiger. 1976. The Notions of Consistency and Predicate Locks in a Database System. *Commun. ACM* 19, 11 (1976), 624–633. https://doi.org/10.1145/360363.360369

[8] Aptos Foundation. 2023. *Aptos Core*. https://github.com/aptos-labs/aptos-core

[9] Rati Gelashvili, Alexander Spiegelman, Zhuolun Xiang, George Danezis, Zekun Li, Dahlia Malkhi, Yu Xia, and Runtian Zhou. 2023. Block-STM: Scaling Blockchain Execution by Turning Ordering Curse to a Performance Blessing. In *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming, PPoPP 2023, Montreal, QC, Canada, 25 February 2023 - 1 March 2023*, Maryam Mehri Dehnavi, Milind Kulkarni, and Sriram Krishnamoorthy (Eds.). ACM, 232–244. https://doi.org/10.1145/3572848.3577524

[10] Masoomeh Javidi Kishi, Sebastiano Peluso, Henry F. Korth, and Roberto Palmieri. 2019. SSS: Scalable Key-Value Store with External Consistent and Abort-free Read-only Transactions. In *39th IEEE International Conference on Distributed Computing Systems, ICDCS 2019, Dallas, TX, USA, July 7-10, 2019*. IEEE, 589–600. https://doi.org/10.1109/ICDCS.2019.00065

[11] Asia Krishna Srinivasan and Pacific Department Director. 2022. Opening remarks at peer-learning series on Digital Money/Technology: Central Bank Digital Currency and the case of China. https://www.imf.org/en/News/Articles/2022/07/07/sp070722-central-bank-digital-currency-and-the-case-of-china#.ZD2fh81KeHA.link

[12] David Mazieres. 2015. The stellar consensus protocol: A federated model for internet-level consensus. *Stellar Development Foundation* 32 (2015), 1–45.

[13] Satoshi Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System. *Bitcoin.–URL: https://bitcoin. org/bitcoin. pdf* (2008).

[14] Guna Prasaad, Alvin Cheung, and Dan Suciu. 2020. Handling Highly Contended OLTP Workloads Using Fast Dynamic Partitioning. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). ACM, 527–542. https://doi.org/10.1145/3318464.3389764

[15] Hany E. Ramadan, Christopher J. Rossbach, and Emmett Witchel. 2008. Dependence-aware transactional memory for increased concurrency. In *2008 41st IEEE/ACM International Symposium on Microarchitecture*. 246–257. https://doi.org/10.1109/MICRO.2008.4771795

[16] Arun Raman, Hanjun Kim, Thomas R Mason, Thomas B Jablin, and David I August. 2010. Speculative parallelization using software multi-threaded transactions. In *Proceedings of the fifteenth International Conference on Architectural support for programming languages and operating systems*. 65–76.

[17] Geoffrey Ramseyer, Ashish Goel, and David Mazières. 2023. SPEEDEX: A Scalable, Parallelizable, and Economically Efficient Decentralized EXchange. arXiv:2111.02719 [cs.DC]

[18] Mohamed M. Saad, Masoomeh Javidi Kishi, Shihao Jing, Sandeep Hans, and Roberto Palmieri. 2019. Processing transactions in a predefined order. In *Proceedings of the 24th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP 2019, Washington, DC, USA, February 16-20, 2019*, Jeffrey K. Hollingsworth and Idit Keidar (Eds.). ACM, 120–132. https://doi.org/10.1145/3293883.3295730

[19] Avi Silberschatz, Henry F. Korth, and S. Sudarshan. 2020. *Database System Concepts, Seventh Edition*. McGraw-Hill Book Company. https://www.db-book.com/

[20] Alexander Thomson, Thaddeus Diamond, Shu-Chun Weng, Kun Ren, Philip Shao, and Daniel J Abadi. 2012. Calvin: fast distributed transactions for partitioned database systems. In *Proceedings of the 2012 ACM SIGMOD international conference on management of data*. 1–12.

[21] Gavin Wood. [n. d.]. Ethereum: A secure decentralised generalised transaction ledger. ([n. d.]). https://gavwood.com/paper.pdf