Unleashing Multicore Strength for Efficient Execution of Blockchain Transactions

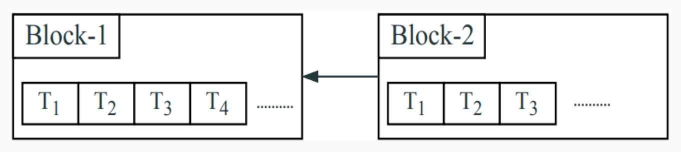
Ankit Ravish Akshay Tejwani **Piduguralla Manaswini**
Sathya Peri

Indian Institute of Technology Hyderabad

21st International Conference on Distributed Computing and Intelligent Technology
2025

Introduction

- Blockchain is a distributed, decentralized database or ledger of records



- Block creator adds the block to the blockchain
- Validators validate the blocks added to the blockchain
- Example: Bitcoin[1], Ethereum[2], Hyperledger Sawtooth[3].

[1] *Bitcoin* (n.d.). Bitcoin.org. URL: https://bitcoin.org/.
[2] *Ethereum* (n.d.). Ethereum.org. URL: https://ethereum.org/.
[3] *Hyperledger Sawtooth Whitepaper* (n.d.). https://8112310.fs1.hubspotusercontent-na1.net/hubfs/8112310/Hyperledger/Offers/Hyperledger_Sawtooth_WhitePaper.pdf.

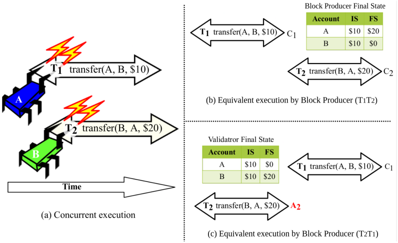
Motivation

- Blockchain adoption faces scalability issues due to consensus and sequential SCT validation[4]
- Parallel execution improves throughput and responsiveness
- Parallel Execution Challenges:
  - Conflicts between block producer and validators
  - Potential state discrepancies leading to block rejection
- **Proposed Solution**: Robust mechanisms to synchronize transactions

[4] Thomas Dickerson et al. (2017). "Adding Concurrency to Smart Contracts". In: PODC '17. Washington, DC, USA: ACM, pp. 303–312. ISBN: 978-1-4503-4992-5.

Parallel Execution Challenges

Validator nodes may incorrectly reject a valid block proposed by the block producer. We call such error as **False Block Rejection (FBR)** error[5],[6].



[5] Thomas Dickerson et al. (2017). "Adding Concurrency to Smart Contracts". In: PODC '17. Washington, DC, USA: ACM, pp. 303–312. ISBN: 978-1-4503-4992-5.

[6] Parwat Singh Anjana et al. (2020). "Efficient Concurrent Execution of Smart Contracts in Blockchains Using Object-Based Transactional Memory". In: *Networked Systems - 8th International Conference, NETYS*. vol. 12129. Springer, pp. 77–93.

Proposed Framework

Multi-Bin Parallel Scheduler (MBPS)

- **MBPS Framework**: Leverages multicore systems for parallel SCT execution
- **Three Variants:**
  1. Standard MBPS: Barrier-based.
  2. Assisted MBPS: Barrier-based with helper threads.
  3. Lockfree MBPS: No barriers, uses atomic operations.

Comparison:

Framework	Barrier Free	Helper Threads
Standard MBPS	$\times$	$\times$
Assisted MBPS	$\times$	$\checkmark$
Lockfree MBPS	$\checkmark$	$\checkmark$

MBPS Framework Architecture

- Key Stages:
  - **Conflict Detection**: Identifies transaction conflicts (read-write, write-write)
  - **Bin Creation**: Groups non-conflicting transactions for parallel execution
  - **Execution**: Executes bins sequentially while maintaining parallelism within bins

MBPS Framework Architecture

Conflict Detection

- T1, T2, T3. . . ,T7 are the list of transactions in the block
- Data items accessed by these transactions - A, B, C, D, E, F, G
  - T1 reads A and writes B
  - T2 reads C and writes A
  - T3 writes C and reads D
  - T4 reads E
  - T5 writes D,G and reads A,F
  - T6 writes F
  - T7 writes G

conflict[T1]= [ ]

conflict[T2]= [T1]

conflict[T3]= [T2]

conflict[T4]= [ ]

conflict[T5]= [T2,T3]

conflict[T6]= [T5]

conflict[T7]= [T5]

MBPS Framework Architecture

- Key Stages:
  - **Conflict Detection**: Identifies transaction conflicts (read-write, write-write)
  - **Bin Creation**: Groups non-conflicting transactions for parallel execution

MBPS Framework Architecture

Bin Assignment

conflict[T1]= [ ]	
conflict[T2]= [T1]	bin[1]= [T1,T4]
conflict[T3]= [T2]	bin[2]= [T2]
conflict[T4]= [ ]	bin[3]= [T3]
conflict[T5]= [T2,T3]	bin[4]= [T5]
conflict[T6]= [T5]	bin[5]= [T6,T7]
conflict[T7]= [T5]	

MBPS Framework Architecture

- Key Stages:
  - **Conflict Detection**: Identifies transaction conflicts (read-write, write-write)
  - **Bin Creation**: Groups non-conflicting transactions for parallel execution
  - **Execution**: Executes bins sequentially while maintaining parallelism within bins

MBPS Framework Architecture

- Key Stages:
  - **Conflict Detection**: Identifies transaction conflicts (read-write, write-write)
  - **Bin Creation**: Groups non-conflicting transactions for parallel execution
  - **Execution**: Executes bins sequentially while maintaining parallelism within bins

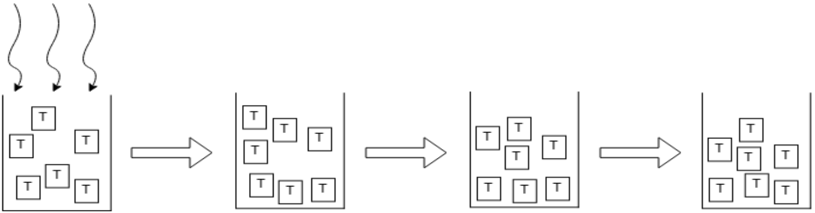


Figure: Multi Binning Execution

MBPS Framework Architecture

- Key Stages:
  - **Conflict Detection**: Identifies transaction conflicts (read-write, write-write)
  - **Bin Creation**: Groups non-conflicting transactions for parallel execution
  - **Execution**: Executes bins sequentially while maintaining parallelism within bins

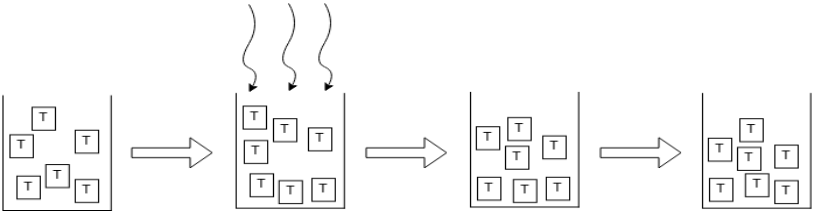


Figure: Multi Binning Execution

MBPS Framework Architecture

- Key Stages:
  - **Conflict Detection**: Identifies transaction conflicts (read-write, write-write)
  - **Bin Creation**: Groups non-conflicting transactions for parallel execution
  - **Execution**: Executes bins sequentially while maintaining parallelism within bins

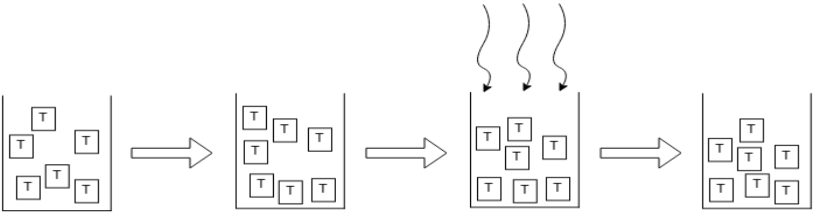


Figure: Multi Binning Execution

MBPS Framework Architecture

- Key Stages:
  - **Conflict Detection**: Identifies transaction conflicts (read-write, write-write)
  - **Bin Creation**: Groups non-conflicting transactions for parallel execution
  - **Execution**: Executes bins sequentially while maintaining parallelism within bins

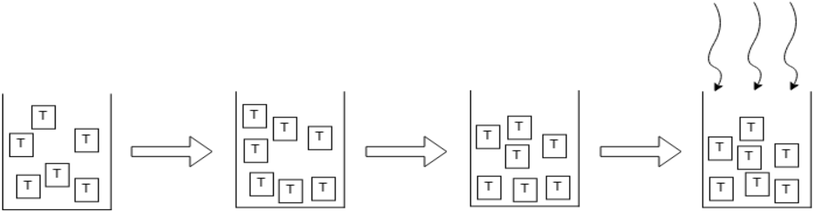


Figure: Multi Binning Execution

MBPS Framework Architecture

- **Standard MBPS:**
  - Uses barriers for controlled synchronization
  - Phases: Conflict Set Identification and Bin Number Assignment
  - Efficient but suffers from synchronization overhead and threads latency
- **Assisted MBPS:**
  - Enhances Standard MBPS with helper threads for optimized execution
  - Helps when some threads are slow or stuck
  - Improves efficiency, but still uses barriers
- **Lockfree MBPS:**
  - No barriers or locks, so threads don't have to wait
  - Uses atomic operations and helper threads for faster parallel execution
  - Threads crash tolerant

Experimental Evaluation

- 2-socket Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60GHz
- 56 threads (14 cores per socket and 2 threads per core)
- Ubuntu 18.04.6 LTS
- Varying transactions in a block
- Varying threads
- Varying conflict%
- C++ language
- Tested on Hyperledger Sawtooth Blockchain

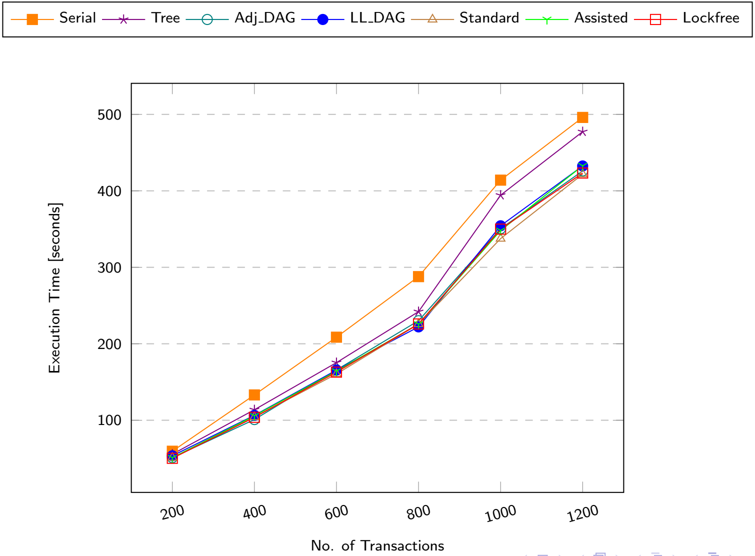
Experimental Evaluation

- Three distinct experiments to assess the performance:
  - **Baseline Performance**
  - **Threads Latency Analysis**
  - **Threads Crash Analysis**

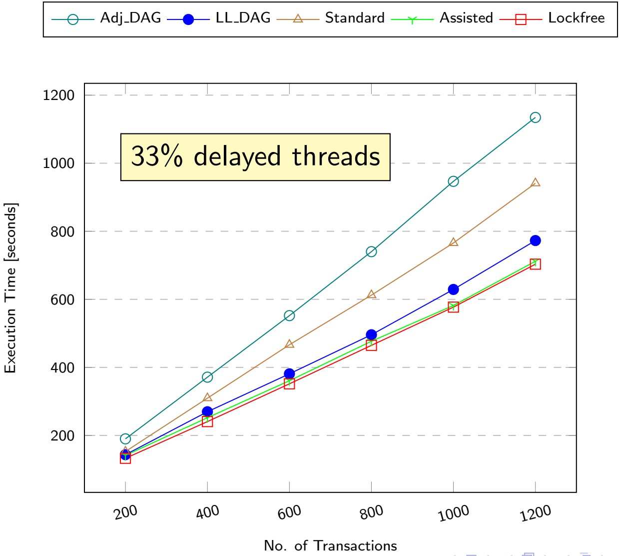
- Three distinct conflict parameters (CP)[7] to assess the performance:
  - **CP1:** Percentage of transactions having at least one dependency.
  - **CP2:** Percentage of dependent transactions in relation to the total number of transactions.
  - **CP3:** Percentage of disjoint transactions relative to the total number of transactions.

[7] Manaswini Piduguralla et al. (2023). "DAG-Based Efficient Parallel Scheduler for Blockchains: Hyperledger Sawtooth as a Case Study". In: *Euro-Par 2023: Parallel Processing*. Ed. by José Cano et al. Cham: Springer Nature Switzerland, pp. 184–198. ISBN: 978-3-031-39698-4.

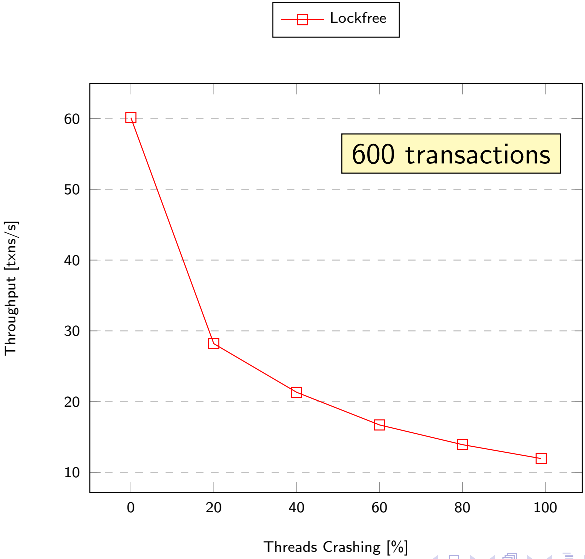
Baseline Performance - CP1



Threads Latency - CP2



Threads Crash - CP3



Conclusion and Future Work

Conclusion:

- Proposed MBPS framework (Standard, Assisted, Lockfree) to parallelize blockchain transactions on multicore systems
- Achieved significant performance gains in throughput on Hyperledger Sawtooth

Future Work:

- Furthur study the designed algorithms under varying conditions
- Optimize conflict detection and bin assignment for efficiency
- Develop the lock-free proof

Thank You

Piduguralla Manaswini
Email: cs20resch11007@iith.ac.in

Ankit Ravish
Email: cs21resch11014@iith.ac.in