CCTS PROJECT REPORT

RISHABH JAIN : CS23MTECH12007
SHALU PRATHMESH RAJIV : CS23MTECH12008

INTRODUCTION

This project focuses on simulating and optimizing the execution of blockchain-like transactions under constrained time and conflict conditions. The objective is to maximize the total gas fee collected by efficiently scheduling transaction execution. Two strategies are implemented: sequential execution, where transactions are processed one by one, and parallel execution, which leverages graph coloring to identify non-conflicting groups that can be run concurrently. A probabilistic conflict graph is built based on transaction read-write dependencies and a given conflict percentage. By comparing the performance of these two approaches, the project highlights the benefits of parallelism in throughput-oriented blockchain systems.

PROBLEM STATEMENT

a) In blockchain systems, transactions often have dependencies based on shared read and write operations, which can lead to execution conflicts. Each transaction in the pool is associated with three key attributes:

- **Execution Time**: Time required to process the transaction.
- **Gas Fee**: The reward earned for successfully executing the transaction.
- **Dependencies**: Defined by the transaction's Read-Set and Write-Set, indicating which data items it accesses or modifies.

b) The goal is to select and execute a subset of these transactions such that:

- The total execution time does not exceed a predefined time limit.
- The number of successfully executed transactions is maximized as compared to serial
- The total gas fee collected from these executions is maximized.

To achieve this, both **sequential** and **parallel execution strategies** are explored. In the parallel approach, concurrency is leveraged by detecting conflicts and grouping non-conflicting transactions using graph coloring. This simulation aims to demonstrate how optimized scheduling and parallelism can enhance blockchain performance by increasing throughput and reducing idle time due to conflicts.

APPROACH

1) Sequential Execution

- Sorted by gas fee.
- Simulated one-by-one using sleep delay.
- Stops execution once the time limit is reached.

2) Parallel Execution (Graph Coloring)

- Conflict graph built probabilistically based on overlap and conflict percentage.
- Coloring to form independent transaction groups.
- Groups sorted by total gas fee and greedily selected.
- Transactions executed in parallel using multithreading

ALGORITHMS

1) Serial Execution algorithm

Input: threads, conflictPercentage, totalTime, transactions[]
Output: TotalGasFeeCollected, TransactionsExecuted, sequential_result.txt

1. ParseInput("input.txt")
    Read: threads, conflictPercentage, totalTime, numTransactions
    For each transaction:
    Read: id, readSet, writeSet, gasFee
    Store in Transaction struct

2. SortTransactionsByGasFee(transactions)
    Sort the list of transactions in descending order of gasFee

3. InitializeExecutionTimer()
    Set totalGasFee = 0
    Set transactionsExecuted = 0
    Set sleepDuration = 0.3 seconds
    Start timer: start = current time

4. SequentiallyExecuteTransactions()
    For each transaction tx in sorted list:
    elapsedTime = current time - start
    If elapsedTime + sleepDuration > totalTime:
    Break // not enough time left for another transaction

    Sleep for 0.3 seconds to simulate execution
    Add tx.gasFee to totalGasFee
    Increment transactionsExecuted
    Log tx.id and tx.gasFee

5. LogExecutionSummary("sequential_result.txt")
    Write:
    Total transactions executed
    Total gas fee collected
    Total time used
    Time limit provided
EndAlgorithm

2) Parallel execution using graph coloring

Input: threads, conflictPercentage, totalTime, transactions[]
Output: TotalGasFee, ExecutedTransactions, result.txt

1. ParseInput("input.txt")
    Read: threads, conflictPercentage, totalTime, numTransactions
    For each transaction:
    Read: id, readSet, writeSet, gasFee
    Store in Transaction struct

2. BuildConflictGraph(transactions, conflictPercentage)
    Initialize graph as adjacency list
    For i from 0 to n-1:
    For j from i+1 to n-1:
    If tx[i] and tx[j] conflict:
        With probability ≤ conflictPercentage:
        Add edge between i and j
    Return conflictGraph

3. GraphColoring(conflictGraph)
    Initialize result[n] = -1 (colors), used[n] = false
    For each node u in 0 to n-1:
    Mark colors used by u's neighbors
    Assign lowest unused color to u
    Return result // colors assigned to transactions

4. GroupByColor(transactions, colors)
    For each transaction i:
    Add i to colorGroup[colors[i]]
    Add tx[i].gasFee to colorGasFee[colors[i]]
    For each color:
    timeNeeded = ceil(groupSize / threads)
    Store ColorInfo: color, gasFee, timeNeeded, txIDs
    Return colorInfos

5. SelectColorGroups(colorInfos, totalTime - graphBuildTime)
    Sort colorInfos by descending gasFee
    Initialize: totalFee = 0, timeUsed = 0
    For each group in colorInfos:

If timeUsed + group.timeNeeded ≤ remainingTime:
Mark group as selected
Add group.gasFee to totalFee
Add group.txIDs to selectedTransactions
Update timeUsed
Return totalFee, selectedTransactions, executedColorGroups

6. SimulateExecution(selectedTransactions)
For each txID in selectedTransactions:
Sleep for 300ms
Print "Executed transaction ID: tx[txID].id"

7. WriteOutput("result.txt")
Write: Total gas fee collected
Write: Graph build time
Write: Remaining time
For each color group:
List all transactions
Indicate if executed or not
Include total gas fee for the group

EndAlgorithm

EXPERIMENTS

1) In my first experiment I am taking the uniform distribution and sleep as 0.3 second.

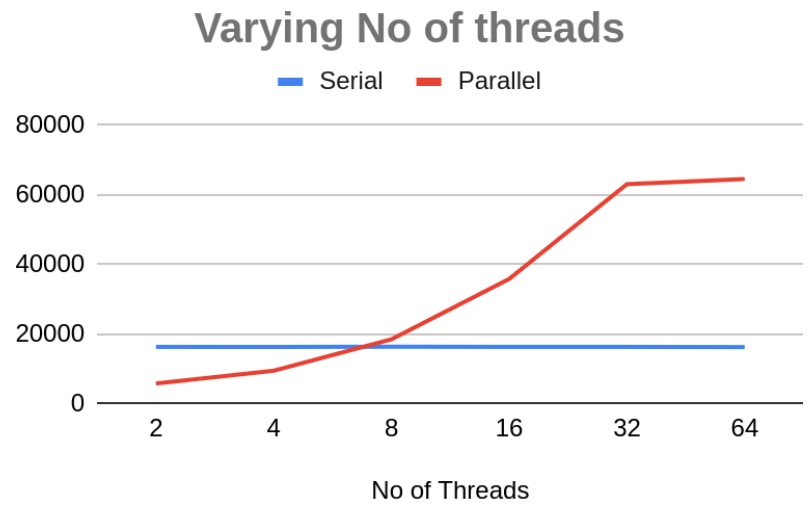
Here i have varied gas fees from 10 to 1000 and using uniform distribution

Given constraints are :

1) Number of threads
2) Given block creation time
3) No of transactions
4) Sleep given

Experiment 1 : Varying the number of threads

No of Threads	Serial	Parallel
2	16294	5783
4	16288	9443
8	16354	18447
16	16289	35781
32	16307	63007
64	16235	64478

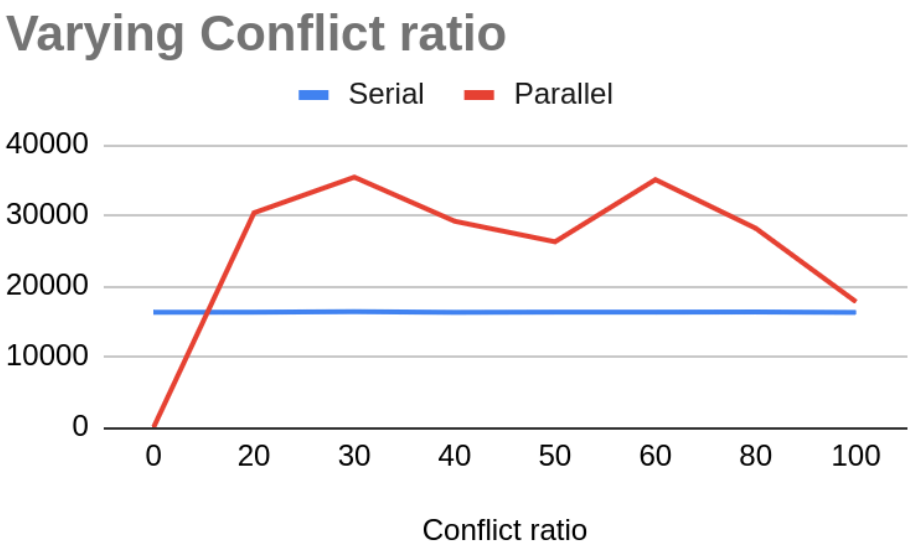


Given : no of transactions taken as 1500
Conflict dependency : 30%

Result : Returning maximum gas fee for both serial and parallel when varying number of threads

Experiment 2 : Varying conflict ratio

Conflict ratio	Serial	Parallel
0	16288	0
20	16306	30342
30	16379	35381
40	16278	29146
50	16308	26236
60	16313	35026
80	16335	28140
100	16264	17723

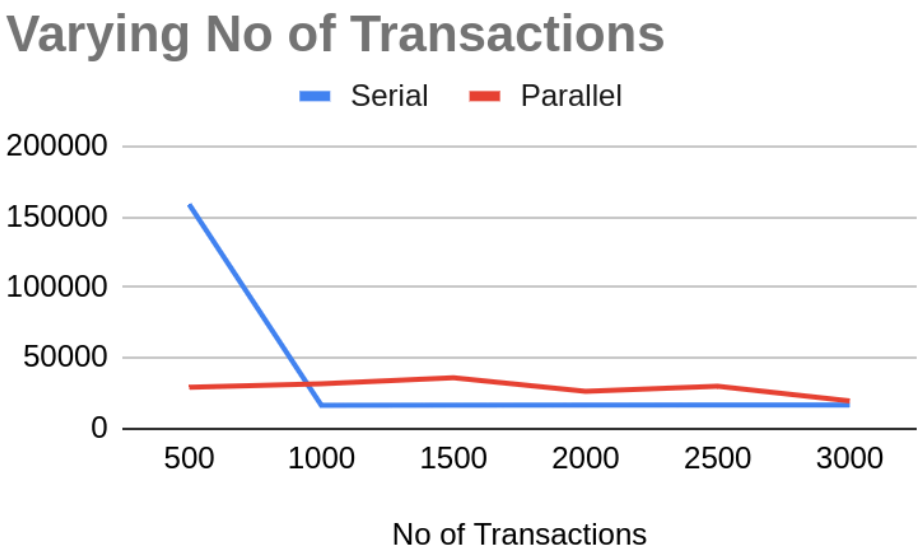


Given : No of threads as 16
No of transactions : 1500

Output : Returning the maximum gas fees

Experiment 3 : Varying number of transactions

No of Transactions	Serial	Parallel
500	158892	29021
1000	16161	31480
1500	16289	35781
2000	16322	26150
2500	16379	29731
3000	16389	19279

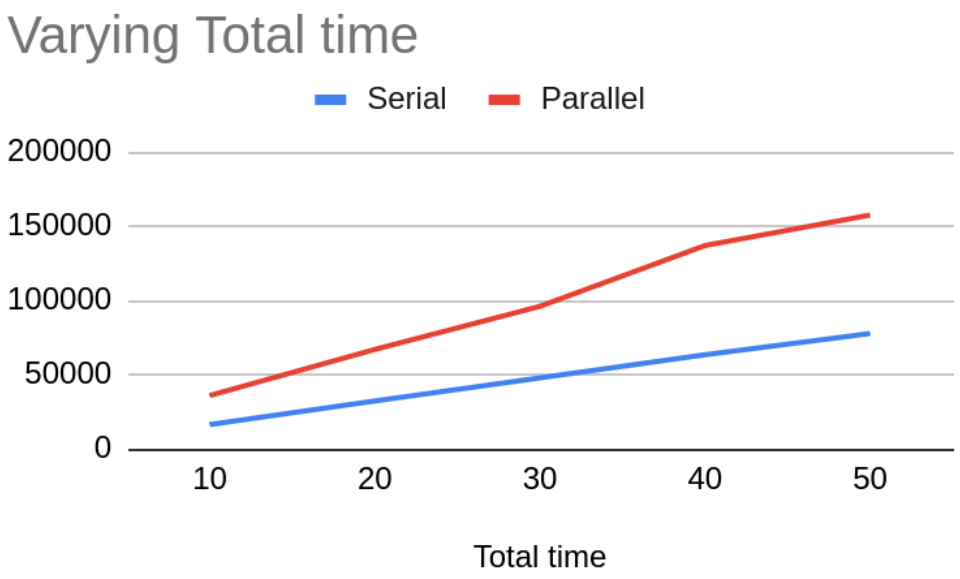


Given : No of threads : 16
Conflict ratio taken as 30 %

Output : Returning maximum gas fees for both serial and parallel execution when varying number of transactions

Experiment 4 : Varying total time

Total time	Serial	Parallel
10	16289	35784
20	32188	66806
30	47737	95870
40	63337	136829
50	77643	157375



Given : No of threads as 16
Conflict ratio taken as 30%
No of transactions taken as 1500

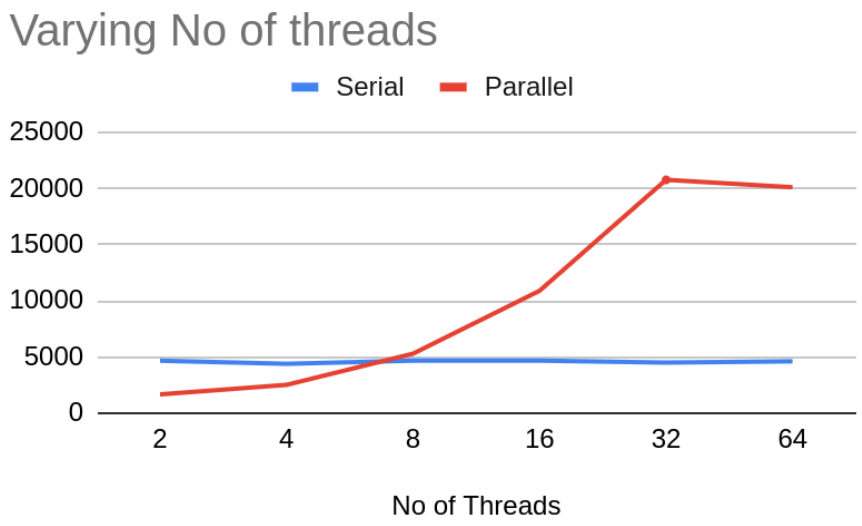
Output : Returning the maximum gas fees for the serial and the parallel execution

Bell Distribution

Now here bell distribution means i have generated the gas fee for each transaction following bell distribution and i will be showing the same results as shown above just changed the distribution here

Experiment 5 : Varying the number of threads

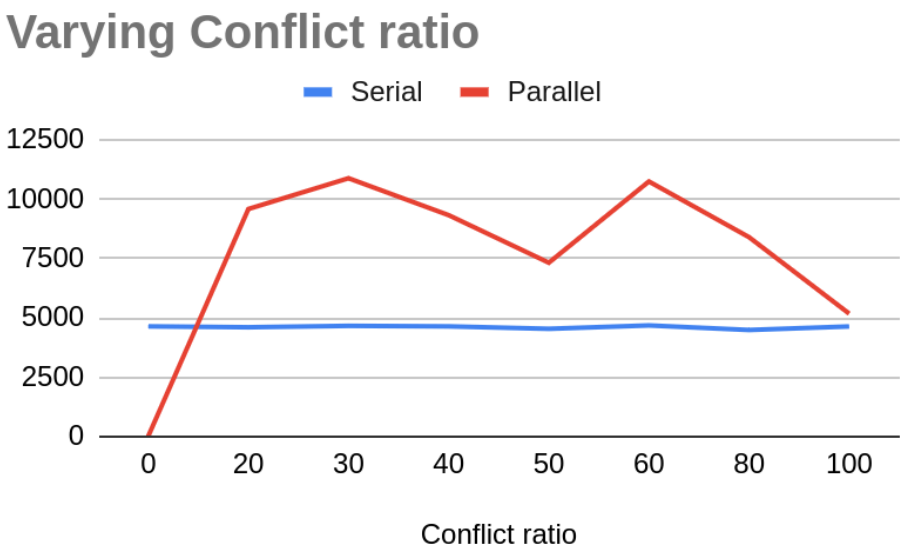
No of Threads	Serial	Parallel
2	4655	1647
4	4373	2494
8	4662	5279
16	4661	10895
32	4491	20798
64	4599	20132



Given : no of transactions taken as 1500
Conflict dependency : 30%

Experiment 6 : Varying conflict ratio

Conflict ratio	Serial	Parallel
0	4642	0
20	4602	9596
30	4661	10895
40	4642	9334
50	4537	7328
60	4681	10759
80	4488	8404
100	4636	5176

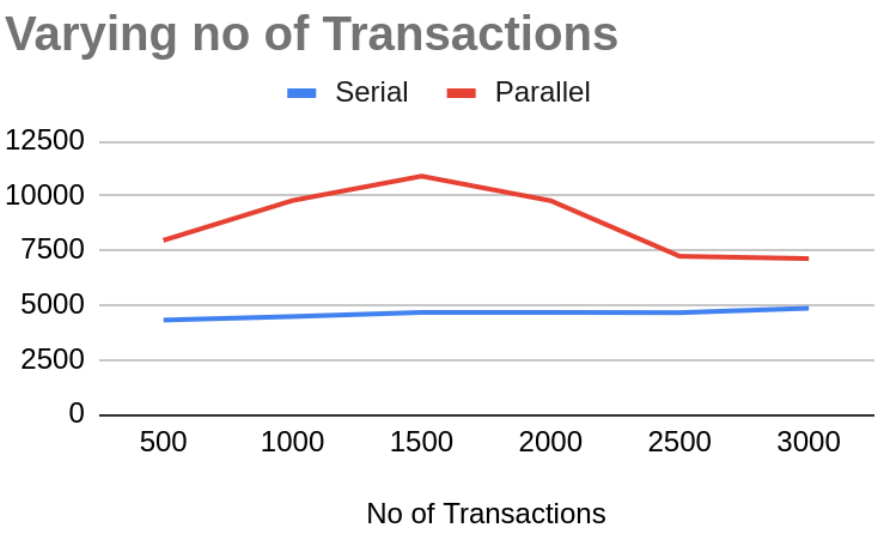


Given : No of threads as 16
No of transactions : 1500

Output : Returning the maximum gas fees

Experiment 7 : Varying number of transactions

No of Transactions	Serial	Parallel
500	4315	7957
1000	4476	9783
1500	4662	10895
2000	4664	9771
2500	4653	7230
3000	4855	7121

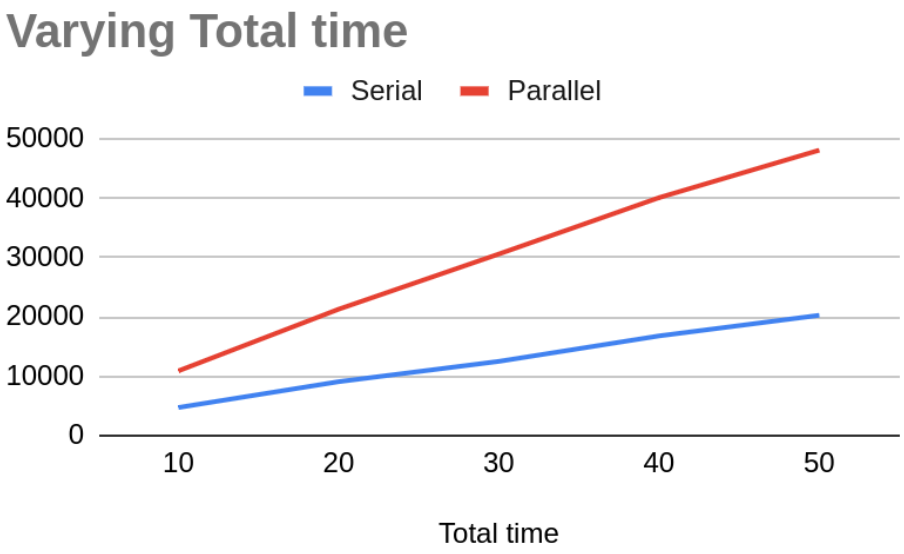


Given : No of threads : 16
Conflict ratio taken as 30 %

Output : Returning maximum gas fees for both serial and parallel execution when varying number of transactions

Experiment 8 : Varying total time

Total time	Serial	Parallel
10	4661	10895
20	9020	21314
30	12455	30637
40	16757	40166
50	20241	48186



Given : No of threads as 16
Conflict ratio taken as 30%
No of transactions taken as 1500

Output : Returning the maximum gas fees for the serial and the parallel execution

Conclusion

1) As discussed in our problem statement we were able to return the maximum number of gas fees i.e we are getting the maximum gas fee for our optimal approach as compared to serial execution.

2) Also as through experiments i observe that i was getting maximum number of transactions being executed for parallel as compared to serial which was our secondary goal.

3) Here i have varied the number of threads , number of transactions , conflict dependency and also the block creation time i.e the total time