**Group C: MINI PROJECT**

**Problem Statement:** Develop a Blockchain based application dApp (de- centralized app) for evoting system.

**Theory:**

Blockchain is a technology that is rapidly gaining momentum in era of industry 4.0. With high security and transparency provisions, it is being widely used in supply chain management systems, healthcare, payments, business, IoT, voting systems, etc. Why do we need it? Current voting systems like ballot box voting or electronic voting suffer from various security threats such as DDoS attacks, polling booth capturing, vote alteration and manipulation, malware attacks, etc, and also require huge amounts of paperwork, human resources, and time. This creates a sense of distrust among existing systems.

Some of the disadvantages are:

- Long Queues during elections.
- Security Breaches like data leaks, vote tampering.
- Lot of paperwork involved, hence less eco-friendly and time-consuming.
- Difficult for differently-abled voters to reach polling booth.
- Cost of expenditure on elections is high.

**Solution:** Using blockchain, voting process can be made more secure, transparent, immutable, and reliable. How? Let‘s take an example. Suppose you are an eligible voter who goes to polling booth and cast vote using EVM (Electronic Voting Machine). But since it‘s a circuitry after all and if someone tampers with microchip, you may never know that did your vote reach to person for whom you voted or was diverted into another candidate‘s account? Since there‘s no tracing back of your vote. But, if you use blockchain- it stores everything as a transaction that will be explained soon below; and hence gives you a receipt of your vote (in a form of a transaction ID) and you can use it to ensure that your vote has been counted securely.

Now suppose a digital voting system (website/app) has been launched to digitize process and all confidential data is stored on a single admin server/machine, if someone tries to hack it or snoop over it, he/she can change candidate‘s vote count- from 2 to 22! You may never know that hacker installs malware or performs clickjacking attacks to steal or negate your vote or simply attacks central server.

To avoid this, if system is integrated with blockchain- a special property called immutability protects system. Consider SQL, PHP, or any other traditional database systems. You can insert, update, or delete votes. But in a blockchain you can just insert data but cannot update or delete. Hence when you insert something, it stays there forever and no one can manipulate it- Thus name immutable ledger. But Building a blockchain system is not enough. It should be decentralized i.e if one server goes down or something happens on a particular node, other nodes can function normally and do not have to wait for victim node‘s recovery.

So a gist of advantages are listed below:

• You can vote anytime/anywhere (During Pandemics like COVID-19 where it‘s impossible to hold elections physically

• Secure

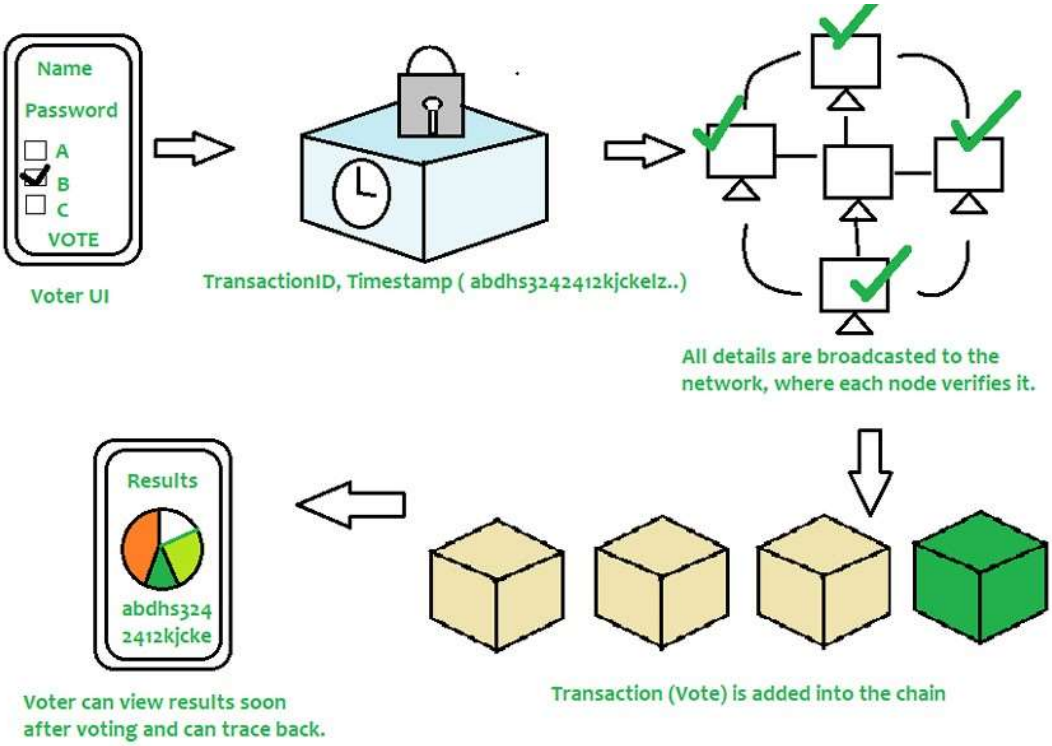
• Immutable

• Faster

• Transparent

**Let’s visualize process**

It is always interesting to learn things if it‘s visually explained. Hence diagram given below explains how the blockchain voting works.




According to above diagram, voter needs to enter his/her credentials in order to vote. All data is then encrypted and stored as a transaction. This transaction is then broadcasted to every node in network, which in turn is then verified. If network approves transaction, it is stored in a block and added to chain. Note that once a block is added into chain, it stays there forever and can‘t be updated. Users can now see results and also trace back transaction if they want.

Since current voting systems don‘t suffice to security needs of modern generation, there is a need to build a system that leverages security, convenience, and trust involved in voting process. Hence voting systems make use of Blockchain technology to add an extra layer of

security and encourage people to vote from any time, anywhere without any hassle and makes voting process more cost-effective and time-saving.

**Conclusion:**

Hence we have successfully made this mini project.

**Group A : MINI PROJECT**

**Title:** Write a program to implement matrix multiplication. Also implement multithreaded matrix multiplication with either one thread per row or one thread per cell. Analyze and compare their performance.

**Objective of the Assignment**: Students will be able to implement and analyze matrix multiplication and multithreaded matrix multiplication with either one thread per row or one thread per cell.

**Prerequisite:**

1. Basic Knowledge of python or Java
2. Concept of Matrix Multiplication

**Theory:**

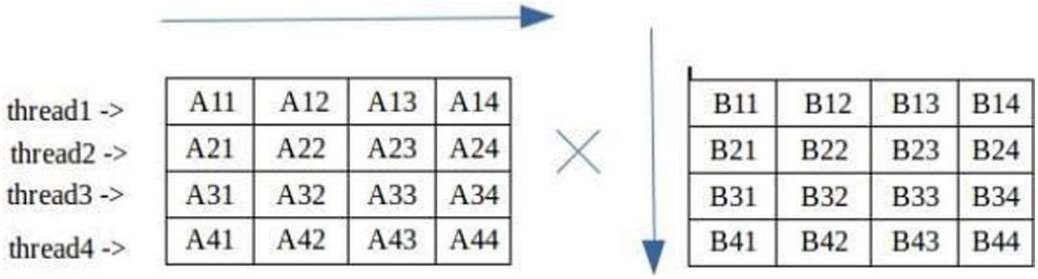
Multiplication of matrix does take time surely.

Time complexity of matrix multiplication is O(n^3) using normal matrix multiplication. And Strassen algorithm improves it and its time complexity is O(n^(2.8074)). But, Is there any way to improve the performance of matrix multiplication using the normalmethod.

Multi-threading can be done to improve it.In multi-threading, instead of utilizing a single core of your processor, we utilizes all or more core to solve the problem. We create different threads, each thread evaluating some part of matrix multiplication.

Depending upon the number of cores your processor has, you can create the number of threads required. Although you can create as many threads as you need, a better way is to create each thread for one core.

In second approach,we create a separate thread for each element in resultant matrix. Using pthread_exit() we return computed value from each thread which is collected by pthread_join().This approach does not make use of any global variables.




To compare the performance of matrix multiplication with one thread per row and one thread per cell, we'll analyze the execution times and consider factors like matrix size and the number of CPU threads available. In general, the choice of which method is more efficient will depend on the specific use case and hardware configuration. Let's analyze the performance:

**1. Matrix Size (m, n, p):** In this example, we used matrices of size 1000x1000 for A and B. Smaller matrices may not demonstrate significant performance differences, while larger matrices may show more noticeable speedups from parallelization.

**2. Multithreading Overhead:** The use of multiple threads introduces overhead in terms of thread creation, synchronization, and context switching. For smaller matrices, this overhead

can dominate, and one thread per cell may perform better. For larger matrices, the actual matrix multiplication work may dominate, and one thread per row may perform better.

**3. Number of CPU Threads:** The number of available CPU threads can affect the performance. If there are fewer threads than rows/columns, it might not make sense to use one thread per row or cell. On a multi-core CPU, if you have many threads available, parallelization can be more beneficial.

**4. Cache and Memory:** The efficiency of cache usage can play a role in performance. One thread per row may perform better when it utilizes the cache more effectively.

**5. Optimized Libraries:** Specialized libraries like NumPy have highly optimized matrix multiplication routines that can outperform custom multithreaded implementations.

**6. Hardware:** The performance can vary based on the specific hardware and CPU architecture.

**7. Implementation Overheads:** The specific implementation of the matrix multiplication algorithm and the way it's parallelized can also impact performance. In this example, we used a basic approach.

**8. Load Balancing:** Load balancing is important when using one thread per row. Uneven workloads can lead to inefficient use of CPU resources.

**9. Scaling:** The performance may not scale linearly with the number of threads. Beyond a certain point, adding more threads may not lead to significant improvements due to the limitations of the hardware

**Code :**

```
import numpy as np

import time

import threading


# Function for matrix multiplication

def matrix_multiply(A, B):

    result = np.dot(A, B)

    return result


# Function to multiply a row with the entire matrix

def multiply_row(row_index, A, B, result):

    for col_index in range(B.shape[1]):

        result[row_index][col_index] = sum(A[row_index][k] * B[k][col_index] for k in range(B.shape[0]))
```

```
# Function to multiply a cell
def multiply_cell(i, j, A, B, result):
    result[i][j] = sum(A[i][k] * B[k][j] for k in range(A.shape[1]))

# Main function to execute the matrix multiplication
if __name__ == "__main__":
    # Define matrices A and B
    A = np.array([
        [1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]
    ])
    B = np.array([
        [9, 8, 7],
        [6, 5, 4],
        [3, 2, 1]
    ])

    # Display input matrices
    print("Matrix A:")
    print(A)
    print("\nMatrix B:")
    print(B)

    # Traditional Matrix Multiplication
    start_time = time.time()
    result_standard = matrix_multiply(A, B)
    end_time = time.time()
    print("\nStandard Matrix Multiplication Time:", end_time - start_time)
```

```
print("Result:")
print(result_standard)

# Multithreaded Matrix Multiplication (one thread per row)
result_row = np.zeros((A.shape[0], B.shape[1]))
threads = []
start_time = time.time()
for i in range(A.shape[0]):
    thread = threading.Thread(target=multiply_row, args=(i, A, B, result_row))
    threads.append(thread)
    thread.start()
for thread in threads:
    thread.join()
end_time = time.time()
print("\nMultithreaded (per row) Matrix Multiplication Time:", end_time - start_time)
print("Result:")
print(result_row)

# Multithreaded Matrix Multiplication (one thread per cell)
result_cell = np.zeros((A.shape[0], B.shape[1]))
threads = []
start_time = time.time()
for i in range(A.shape[0]):
    for j in range(B.shape[1]):
        thread = threading.Thread(target=multiply_cell, args=(i, j, A, B, result_cell))
        threads.append(thread)
        thread.start()
for thread in threads:
    thread.join()
end_time = time.time()
```

```
print("\nMultithreaded (per cell) Matrix Multiplication Time:", end_time - start_time)
print("Result:")
print(result_cell)
```

**Output:**

```
Matrix A:
[[1 2 3]
 [4 5 6]
 [7 8 9]]

Matrix B:
[[9 8 7]
 [6 5 4]
 [3 2 1]]

Standard Matrix Multiplication Time: 2.956390380859375e-05
Result:
[[ 30  24  18]
 [ 84  69  54]
 [138 114  90]]

Multithreaded (per row) Matrix Multiplication Time: 0.007740020751953125
Result:
[[ 30.  24.  18.]
 [ 84.  69.  54.]
 [138. 114.  90.]]

Multithreaded (per cell) Matrix Multiplication Time: 0.004076480865478516
Result:
[[ 30.  24.  18.]
 [ 84.  69.  54.]
 [138. 114.  90.]]
```

**Conclusion:** Hence we have successfully completed the implementation of this Mini Project.

## Group B: MINI PROJECT

**Problem Statement: -** Build a machine learning model that predicts the type of people who survived the Titanic shipwreck using passenger data (i.e. name, age, gender, socio-economic class,etc.).

**Objective:** Students should learn to build a machine learning model.

**Theory:** Here's a step-by-step guide on how to approach this problem using Python and some popular libraries:

1. **Data Collection and Understanding:**
   • Start by obtaining the Titanic dataset, which contains passenger information and survival labels. You can find datasets on websites like Kaggle.
2. **Data Pre-processing:**
   • Clean the data by handling missing values, outliers, and redundant features. • Perform feature engineering to create relevant features or transform existing ones.
   • Encode categorical variables into numerical format using techniques like one-hot encoding.
3. **Data Splitting:**
   • Split your dataset into a training set and a test set. This allows you to evaluate your model's performance on unseen data.
4. **Select a Machine Learning Algorithm:**
   • Choose a classification algorithm suitable for this problem. Common choices include Decision Trees, Random Forests, Logistic Regression, Support Vector Machines, or Gradient Boosting.
5. **Model Training:**
   • Fit your chosen algorithm to the training data. The model learns patterns from the data.
6. **Model Evaluation:**
   • Evaluate your model's performance using metrics like accuracy, precision, recall, F1-score, and the ROC-AUC score. Cross-validation can help in assessing how well the model generalizes to new data.
7. **Hyperparameter Tuning:**
   • Experiment with different hyperparameters to optimize your model's performance. Techniques like grid search or random search can be helpful.
8. **Model Interpretation:**
   • Understand the feature importance or coefficients of your model to interpret how different features affect survival.
9. **Prediction:**
   • Use your trained model to make predictions on new, unseen data or the test set.
10. **Post-processing:**
    • You may need to further process the model's output, such as setting a threshold for classification.

**K-Fold Cross Validation:** K-Fold Cross Validation randomly splits the training data into K subsetscalled folds. Let‟s image we would split our data into 4 folds (K = 4). Ourrandom forest model would be trained and evaluated 4 times, using a different fold for evaluation everytime, while it would be trained on the remaining 3 folds. The image below shows the process, using 4 folds (K = 4). Every row represents one training + evaluation process. In the first row, the model get‟s trained on the first, second and third subset and evaluated on the fourth. In the second row, the model get‟s trained on the second, third andfourth subset and

evaluated on the first. K-Fold Cross Validation repeats this process till every fold acted once as an evaluation fold.

**Random Forest :**

What is Random Forest ?

Random Forest is a supervised learning algorithm. Like you can already see from it‟s name, it creates a forest and makes it somehow random. The „forest— it builds, is an ensemble of Decision Trees, most of the time trained with the —bagging‖ method. The general idea of the bagging method is that a combination of learning models increases the overallresult.

To say it in simple words: Random forest builds multiple decision treesand merges them together to get a more accurate and stable prediction One big advantage of random forest is, that it can be used for both classification and regression problems, which form the majority of currentmachine learning systems. With a few exceptions a random-forest classifier has all the hyperparameters of a decision-tree classifier and alsoall the hyperparameters of a bagging classifier, to control the ensemble itself.

The random-forest algorithm brings extra randomness into the model, when it is growing the trees. Instead of searching for the best feature whilesplitting a node, it searches for the best feature among a random subset offeatures. This process creates a wide diversity, which generally results in abetter model. Therefore when you are growing a tree in random forest, only a random subset of the features is considered for splitting a node. Youcan even make trees more random, by using random thresholds on top of it, for each feature rather than searching for the best possible thresholds (like a normal decision tree does).

Below you can see how a random forest would look like with two trees: