# BIG DATA PROCESSING

## TERM PROJECT - PHASE 1

### Implementation of Strassen's Matrix Multiplication Algorithm Using Apache Spark for Large-Scale Distributed Systems


## Group Members:

**Name:** Aditya Patel
**ID:** 202203027

**Name:** Dharmesh Kota
**ID:** 202203038


## Objective

The primary objective of this project is to implement an optimized version of Strassen's Matrix Multiplication algorithm in Apache Spark called **STARK**, aiming to improve computational efficiency in large-scale distributed systems. By integrating Strassen's algorithm, we intend to reduce the time complexity of matrix multiplication from $O(n^3)$ to $O(n^{2.807})$, leveraging Spark's distributed architecture to handle large datasets efficiently. This will outperform the existing Apache Spark methods (like `MLLib` and `Marlin`) for large matrix sizes, thereby reducing execution time and increasing scalability.

## Plan for Implementation (Roadmap)

### Phase 1: Research

- **Benchmark Existing Solutions**: Investigate current Spark-based matrix multiplication techniques, such as MLLib and Marlin, to understand their strengths and weaknesses. This will help us identify areas for improvement.
- **Study Strassen's Algorithm**: Examine how Strassen's recursive algorithm works, specifically its method of reducing the number of multiplications from 8 to 7. Focus on how we can effectively integrate this algorithm into the Spark framework.

### Phase 2: Implementation and Design

- **Design the Block Matrix Structure**: Create a structure for block matrices using Spark RDDs. This structure will allow us to split and recombine matrices efficiently in a distributed setting.
- **Matrix Division**: Implement the division phase, where we recursively split large matrices into smaller blocks. Optimize the size of these blocks and how we partition the matrices to use memory efficiently.
- **Block Multiplication**: Utilize optimized libraries like Breeze for multiplying these blocks. Implement Strassen's method to perform fewer multiplications, taking advantage of its efficiency.
- **Result Combination**: Combine the multiplied blocks back into the final product matrix. Use parallel addition and subtraction to ensure that all operations are performed quickly and accurately.

### Phase 3: Optimization, Testing, and Validation

- **Optimize Communication**: Reduce the amount of data transferred between Spark nodes by improving how matrix blocks are partitioned and tagged.
- **Performance Tuning**: Test the algorithm with different matrix sizes and fine-tune block and partition sizes to reduce execution time. Compare performance with other Spark libraries like MLLib and Marlin.
- **Functional Testing**: Ensure the algorithm works correctly for different matrix sizes, including special cases like non-square matrices.
- **Performance Benchmarking**: Run tests in a distributed setup to measure execution time, memory usage, and scalability, verifying improvements over existing approaches.

## Work Done so far!

We have completed Phase 1 of the project, focusing on research:

- Benchmarking current Spark-based matrix multiplication techniques, such as MLLib and Marlin, to identify areas for improvement.
- Studying Strassen's algorithm, particularly its reduction of multiplications from 8 to 7, and planning its integration into the Spark framework.

With Phase 1 done, we are ready to move forward and begin the implementation of the remaining phases.

## References

Misra, Chandan, Sourangshu Bhattacharya, and Soumya K. Ghosh. "Stark: Fast and Scalable Strassen's Matrix Multiplication using Apache Spark." *IEEE Transactions on Big Data*, 2020.