# Parallelization of Cooley-Tukey FFT Algorithm

## Introduction

I am proposing to work on the Fast Fourier Transform (FFT), a fundamental algorithm used across signal processing, image analysis, and scientific computing. While the Cooley-Tukey FFT algorithm already reduces the computational complexity from $O(N^2)$ to $O(N\log N)$ through its divide-and-conquer approach, I believe we can further optimize its performance through parallelization. With today's increasing data sizes and real-time processing demands, I aim to parallelize this algorithm to take full advantage of modern multi-core CPUs and GPUs.

## Objectives

1. Analyze the Cooley-Tukey FFT algorithm to identify which computationally intensive components are best suited for parallelization
2. Implement parallel versions using:
    - MPI + OpenMP for multi-core CPU acceleration
    - CUDA for maximizing GPU parallelism
3. Benchmark my parallel implementations against the sequential version across various input sizes
4. Study how well my solution scales across different numbers of CPU cores and GPU thread blocks

## Methodology:

1. I'll begin by studying the Cooley-Tukey decomposition stages, particularly focusing on DFT computations and twiddle factor multiplications that can be computed independently.
2. For my initial parallelization strategy:
    - I'll use MPI + OpenMP to parallelize DFT calculations and twiddle factor multiplications on CPU
    - I'll develop CUDA kernels for GPU acceleration, focusing on data-parallel operations
3. I'll set up my development environment using:
    - C++ for high-performance computing
    - OpenMP/OpenMPI for CPU parallelism and CUDA for GPU computation
4. I'll conduct thorough benchmarking by measuring execution time, speedup, and efficiency, comparing my implementation against established FFT libraries like FFTW and cuFFT

## Expected Outcomes

1. Achieve significant speedup compared to the sequential FFT implementation
2. Deliver a flexible FFT implementation that scales efficiently with CPU cores and GPU threads

## Timeline and Milestones

- By March 3rd : Complete literature review, and have serial FFT code ready
- By March 14th : Implement and test my MPI + OpenMP-based parallelization
- By March 28th: Develop the CUDA-based version and optimize memory access patterns
- By April 6th: Conduct benchmarking, performance evaluation, and scalability analysis
- By April 18th: Final project presentation
- By April 30th: Finish final project report and submission