Parallelizing FFT-Based Image Compression

Mohamed Badawy

July 17, 2025

1 Introduction and Problem Statement

This proposal is for implementing a parallel Fast Fourier Transform (FFT) based image compression algorithm using OpenMP. The FFT is one of the most important algorithms in scientific computing. It is used in diverse fields such as signal processing, audio analysis, medical imaging, computer vision, and compression methods. In this project, I will target two problems:

1. Efficient multiplication of large polynomials
2. Image compression via frequency domain filtering

Both applications are computationally intensive, especially for large input sizes or high-resolution images, making them excellent candidates for parallelization.

2 Motivation and Scope

The motivation for this project arose from the need to process large multimedia data efficiently. For example, polynomial multiplication by direct methods takes $O(nm)$ time, where n and m are the degrees of the input polynomials. Using FFT, this time can be reduced to $O(n \log n)$. Similarly, FFT-based image compression allows us to decompose an image into frequency components, compress it by discarding less significant frequencies, and reconstruct it with minimal loss of information.

Image compression is widely used in photo storage, transmission, and multimedia applications to reduce data requirements. The scope of this work covers:

- Parallelizing FFT for 1D and 2D signals using OpenMP.
- Demonstrating polynomial multiplication acceleration.
- Implementing and evaluating FFT-based image compression.
- Measuring the effect of parallelism (number of threads) on performance and compression quality.

3 Background and Existing Techniques

The Discrete Fourier Transform (DFT) converts signals from their original domain (such as time or space) to the frequency domain and back. Computing DFT directly takes $O(n^2)$, but the Fast Fourier Transform algorithm (FFT)—in particular, the Cooley-Tukey algorithm—reduces this to $O(n \log n)$.

The Cooley-Tukey algorithm is a classic divide-and-conquer method that recursively breaks down a DFT of composite size N into multiple smaller DFTs, usually of size $N/2$. It requires input sizes to be a power of two for optimal performance, though other factorizations are possible. One core step is bit-reversal permutation of input indices, followed by iterative "butterfly" operations across stages.

In the domain of image compression, FFT is used to transform images to the frequency domain, apply a threshold to remove low-magnitude frequencies, and convert back, resulting in compressed output.

Traditionally, these algorithms are implemented sequentially. Some high-performance libraries like FFTW use platform-specific optimizations and multi-threading, but often do not expose the parallelism mechanism for educational coding.

4 Programming Model and Choices

This project will be written in C++ and parallelized using OpenMP. OpenMP is well-suited because:

- It allows inserting directives in C++ code to divide loop iterations across multiple CPU cores.
- Shared-memory makes it straightforward to parallelize independent operations, like the butterfly loops in FFT.
- No significant redesign of the algorithms or data structures is required.

Key elements to be parallelized include:

- Bit-reversal permutation: parallel assignment of reordered elements.
- Iterative FFT stages: butterfly computations can be spread across threads at each stage.
- 2D FFT: apply 1D FFT row-wise and column-wise in parallel.
- Thresholding magnitude values: assign zero to outliers in parallel.

OpenCV will be used for image loading and saving, and accuracy will be checked by comparing parallel results with a reference sequential version.

5 Theoretical Complexity and Expected Gain

The sequential FFT reduces DFT time from $O(n^2)$ to $O(n \log n)$ in 1D, and $O(n^2 \log n)$ for typical 2D image data. Parallel FFT can further reduce the wall-clock time by dividing work among p threads, ideally yielding a speedup up to p times for large n. Empirical results (from previous similar work) show measurable speedup for large image sizes and input arrays when increasing threads, though for small sizes or too many threads, the overhead can reduce gains.

6 Algorithms to be Implemented

The key algorithms in this project will be:

1. **Fast Fourier Transform (1D, 2D):** The Cooley-Tukey iterative in-place FFT algorithm, parallelized across outer loops where possible.
2. **Polynomial Multiplication:** Polynomial coefficients are converted to points by FFT, multiplied, and transformed back, all in parallel.
3. **Image Compression:** 2D FFT on grayscale images, magnitude thresholding, and 2D inverse FFT.

7 Potential Challenges

Several roadblocks may arise:

- **Thread overhead:** For low data sizes or too many threads, the management cost may dominate.
- **Race conditions:** Careful design needed to ensure threads never write to the same memory locations simultaneously.
- **Cache efficiency:** 2D operations (especially image transposition) may suffer from poor cache usage in parallel.
- **Numerical stability:** Parallelizing floating-point operations may yield small differences in output.
- **Load imbalance:** Especially in non-power-of-two sizes or irregular data, load should be managed among threads.

8 Expected Outcomes and Evaluation

The project will result in:

- A documented, tested C++ codebase performing FFT-based polynomial multiplication and image compression in parallel.
- Quantitative graphs comparing time and speedup for sequential vs. parallel runs for various thread counts and input sizes.
- Series of compressed images at multiple thresholds, demonstrating the loss-vs-compression trade-off.
- A final report analyzing scaling, efficiency, and quality of parallel solutions.