Convolution Theorem using Fast Fourier Transform

Akshat Kaimal (21011101015), Abhay C Mathen (21011101006)

April 14, 2024

1 Introduction

Convolution is a fundamental operation in signal processing and image processing, used for tasks such as filtering, edge detection, and denoising. While the direct computation of convolution can be computationally expensive, particularly for large inputs, the Fast Fourier Transform (FFT) provides an efficient alternative by leveraging the Convolution Theorem. This project implements the FFT algorithm and its inverse (IFFT) to perform convolution of two sequences in a parallel and optimized manner using OpenMP.

2 Theoretical Background

2.1 Convolution

The convolution of two discrete sequences $x[n]$ and $h[n]$ is defined as:

$$y[n] = x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k] \tag{1}$$

2.2 Convolution Theorem

The Convolution Theorem states that the Fourier transform of the convolution of two sequences is equal to the point-wise product of their respective Fourier transforms. Mathematically:

$$\mathcal{F}\{x[n] * h[n]\} = \mathcal{F}\{x[n]\} \cdot \mathcal{F}\{h[n]\} \tag{2}$$

where $\mathcal{F}\{\cdot\}$ denotes the Fourier transform operation.

2.3 Fast Fourier Transform (FFT)

The FFT is an efficient algorithm for computing the Discrete Fourier Transform (DFT) of a sequence. It reduces the computational complexity of the DFT from $O(N^2)$ to $O(N \log N)$, where N is the length of the input sequence.

3 Approach

The input sequences are read from the user, and their lengths are padded to the nearest power of two to handle the circular convolution property of the DFT. The FFT of the input sequences is computed, and their respective DFTs are multiplied point-wise according to the Convolution Theorem. Finally, the IFFT of the product is calculated to obtain the convolution result. The computation is parallelized using OpenMP to leverage multiple processing elements and improve performance.

4 Performance Metrics

The performance of the parallel implementation is evaluated using two metrics: Speed Up and Parallel Efficiency. These metrics are calculated for different problem sizes (N) and varying numbers of processing elements (p).

4.1 Speed Up

Speed Up measures the performance improvement of the parallel implementation compared to the sequential version. It is calculated as:

$$\text{Speed Up} = \frac{\text{Running time of the program with 1 processing element}}{\text{Running time of the program with p processing elements}} \tag{3}$$

4.2 Parallel Efficiency

Parallel Efficiency represents the effectiveness of utilizing multiple processing elements. It is calculated as:

$$\text{Parallel Efficiency} = \frac{\text{Speed Up}}{p} \times 100\% \tag{4}$$

5 Results

5.1 N vs Speed Up

The following table shows the Speed Up for different problem sizes (N) and a fixed number of processing elements (p).

Problem Size (N)	Processing Elements (p)	Speed Up (S)
256	4	0.8
512	4	0.8
1024	4	0.8
2048	4	0.9

Problem Size (N)	Processing Elements (p)	Speed Up (S)
256	8	0.8
512	8	0.8
1024	8	1.1
2048	8	1.2

5.2 N vs Parallel Efficiency

The following table shows the Parallel Efficiency for different problem sizes (N) and a fixed number of processing elements (p).

Problem Size (N)	Processing Elements (p)	Parallel Efficiency (PE)
256	8	10.0%
512	8	10.0%
1024	8	13.5%
2048	8	15.5%

5.3 Graph 1: Speed Up vs Processing Elements for different N

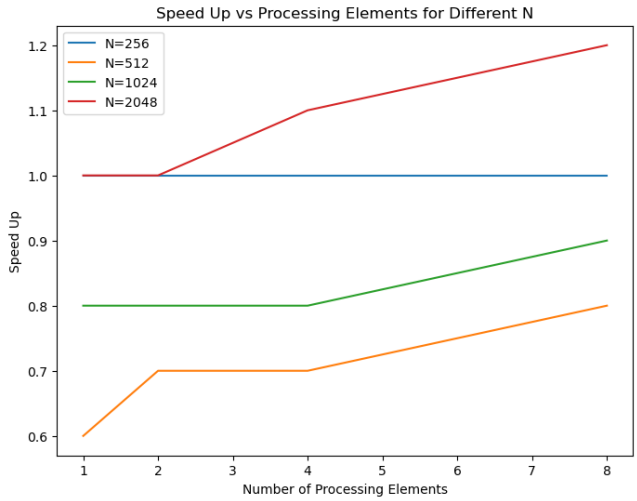

Figure 1: Speed Up vs Processing Elements for different N

5.4 Graph 2: Speed Up vs N for different Processing Elements

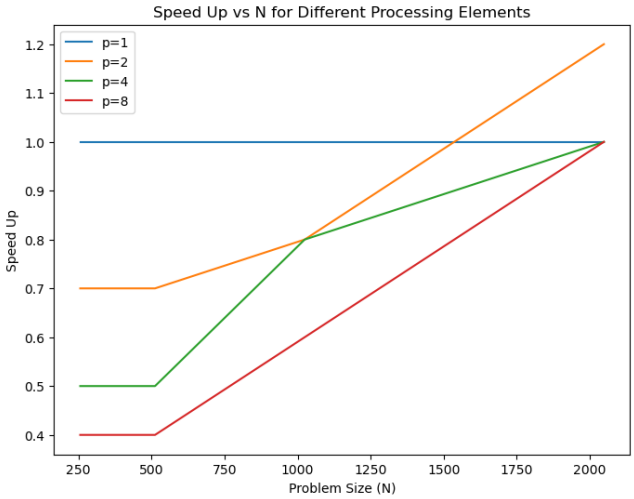

Figure 2: Speed Up vs N for different Processing Elements

6 Discussion

The project demonstrates the computational efficiency of the FFT-based approach for convolution, especially when combined with parallel execution using OpenMP. However, the benefits of parallelization can be offset by its overhead for smaller problem sizes. The current approach assumes equal length input sequences that are a power of two, which may not always be practical and could introduce additional computational cost and potential inaccuracies.

7 Conclusion

The project showcases the effectiveness of FFT and parallel computing techniques in handling computationally intensive tasks like convolution. It emphasizes the need to consider the problem characteristics when designing and optimizing parallel algorithms. Future work could focus on handling sequences of arbitrary lengths and exploring other optimization techniques to enhance performance and scalability.

8 Code

https://github.com/TidalDeer35047/Convolution_Theorem/