Proceedings of Course Project for Parallel Scientific Computing(ID5130)
ID5130 Project
April 27, 2024, Chennai, India

PARALLEL IMPLEMENTATION OF DISCRETE FOURIER TRANSFORMATION

Aniket Khan
ME21B021
Bachelor of Technology
Department of Mechanical Engineering
Indian Institute of Technology, Madras
Chennai 600036
Email: me21b021@smail.iitm.ac.in

ABSTRACT

*The project aims to implement parallel DFT algorithms using **OpenMP** or **MPI** for image processing tasks. It will evaluate performance metrics like **speedup** and **efficiency** while exploring different parallelization strategies and input sizes. The goal is to optimize computational resources for applications in computer vision and medical imaging.*

NOMENCLATURE

$\psi(n,p)$: Speedup
$\sigma(n)$: Inherently sequential Part of the program
$\phi(n)$: Part of the program that can be parallelised
$\kappa(n,p)$: Overheads associated with the parallel program
T_s: Time needed to run the program serially
T_p: Time needed to run the program in parallel
$\eta(n,p)$: Efficiency
n: Program size
p: Number of cores/threads/processors
$f(x,y)$ The reconstructed or orignial image.
$F(u,v)$ The 2D DFT of the input image.
(u,v) The coordinates in the frequency domain.
(x,y) The spatial coordinates in the image domain.
M The number of rows in the image.
N The number of columns in the image.
$e^{-i2\pi(\frac{ux}{M}+\frac{vy}{N})}$ Complex coefficient for DFT.
$e^{i2\pi(\frac{ux}{M}+\frac{vy}{N})}$ Complex coefficient for IDFT.

INTRODUCTION

The aim of this project is to develop a parallel implementation of the Discrete Fourier Transform (DFT) and apply it to image processing tasks. The DFT is a fundamental tool for analyzing and processing signals in the frequency domain, and its parallelization can significantly accelerate computations, particularly for large datasets such as images. The project will focus on implementing the parallel DFT algorithm using OpenMP or MPI for distributed memory systems.

Performance metrics such as speedup, efficiency, and scalability will be evaluated to assess the effectiveness of parallelization. Additionally, the project will investigate the impact of different parallelization strategies and input sizes on performance.

The project will provide insights into the benefits of parallel computing for DFT-based image processing tasks and contribute to the optimization of computational resources in image processing pipelines. The results will be valuable for researchers and practitioners working in fields such as computer vision, remote sensing, and medical imaging.

2D DISCRETE FOURIER TRANSFORMATION

The two-dimensional Discrete Fourier Transform (2D DFT) is a mathematical technique used to analyze the frequency components of two-dimensional signals or images. It extends the concept of the one-dimensional DFT to two dimensions, allowing for the decomposition of a two-dimensional signal into its constituent frequency components in both the horizontal and vertical directions.

The formula for the 2D DFT of an $M \times N$ input image $f(x,y)$ is given by:

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) \cdot e^{-i2\pi(\frac{ux}{M}+\frac{vy}{N})}$$

The 2D DFT operation involves computing the sum of all pixel values in the input image, each multiplied by a complex exponential term corresponding to its spatial frequency. This process is performed for all possible combinations of frequencies u and v within the specified range.

2D INVERSE DISCRETE FOURIER TRANSFORMATION

The Inverse Discrete Fourier Transform (IDFT) is the reverse process of the DFT, used to reconstruct a signal or image from its frequency domain representation. For a given 2D frequency-domain signal $F(u,v)$, the IDFT formula is:

$$f(x,y) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u,v) \cdot e^{i2\pi(\frac{ux}{M}+\frac{vy}{N})}$$

The IDFT operation involves summing up all frequency components weighted by their respective complex exponential terms to reconstruct the original spatial domain signal or image.

IMPLEMENTATION

The 2D DFT can be implemented using direct computation, where the formula is applied directly to each pixel in the input image. Here is a step-by-step overview of how the 2D DFT is implemented using direct computation:

1. **Initialize Result**: Create an empty complex-valued matrix to store the frequency domain representation of the input image.
2. **Compute DFT**: For each frequency (u,v) within the specified range, compute the corresponding complex-valued coefficient $F(u,v)$ using the formula mentioned above.
3. **Result**: The final result is a complex-valued matrix representing the 2D DFT of the input image, with frequency components arranged in the (u,v) domain.

After obtaining the 2D DFT, various operations can be performed in the frequency domain, including filtering, enhancement, compression, and feature extraction. Finally, the processed image can be reconstructed by computing the inverse 2D DFT, which maps the frequency domain representation back to the spatial domain. The time complexity of this process is $O(n^2)$.

FORMULAS

Speedup $\phi(n,p) = \frac{T_s}{T_p}$

Efficiency $\eta(n,p) = \frac{T_s}{T_p * p} = \frac{\phi(n,p)}{p}$

METHOD

For a various image sizes, the Discrete Fourier Transform (DFT) was performed with various thread counts to calculate the speedups and efficiency of parallelization. Prior to computing the DFT, some image processing was conducted using Python. The RGB image was converted to a grayscale image, simplifying the problem by reducing it to pixel intensity values ranging from 0 to 255. This preprocessing step helped streamline the DFT computation process, as it eliminated the need to compute the transform separately for each of the three color channels. Another python script was written to visualise the phase and amplitude decomposition of the original image.

Open MP code description

The basic outline of how the DFT was performed in Open MP is described below:

1. Read the CSV file into a M*N matrix
2. Perform Row Wise 1 dimensional DFT on the matrix
3. Perform Column wise 1 dimension DFT on the matrix
4. The resulting matrix is our transformed matrix.

Therefore, in the code, functions for reading the CSV file, computing the 1D DFT, and the 2D DFT was created. The 2D DFT function invokes the 1D DFT function for each row and then again for each column. Threads are instantiated within this 2D DFT function, and the workload is distributed among them. Additionally, when computing the 1D DFT, the workload is further divided among the threads.

A similar algorithm is implemented for IDFT as well.

Open MPI code description

The basic outline of how the DFT was performed in Open MPI is described below:

1. Read the CSV file into a N*N matrix(N is divisible by number of threads)
2. Perform Row Wise 1 dimensional DFT on the matrix
3. Transpose the matrix
4. Perform Row wise 1 dimension DFT on the matrix once more.
5. The resulting matrix is our transformed matrix.

In the code, the CSV file is first read and stored into an N*N matrix in the root processor, and the size of the matrix is then

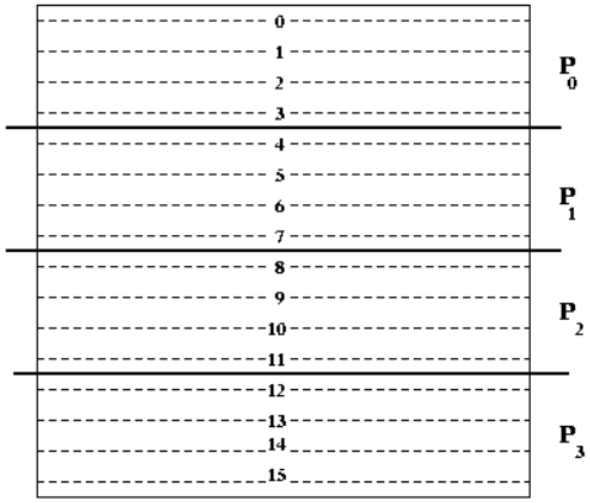

FIGURE 1. ROW WISE ARRAY DECOMPOSITION

broadcasted to all other processors. Following this, row-wise decomposition is conducted using MPI_Scatter(). Subsequently, each processor initiates the 1D DFT computation on their respective datasets. After computation, the results are gathered back in the root processor using MPI_Gather() and transposed. Then, the data is once again decomposed row-wise and sent to the processors for computation. Finally, the transformed image is collected back on the root processor and transposed once more to obtain the final result.

RESULTS

The time needed to execute each code varies from system to system, resulting in different values for each run. Therefore, for each data point, the experiment was repeated 10 times, and the average values were computed. Additionally, the number of threads is limited to 8 due to system constraints.

OPENMP IMPLEMENTATION

For the speedup and efficiency results, an image size of 355x343 was utilized. For scalability analysis, image sizes of 355x343, 720x405, and 720x1280 were employed.

DFT Results

1. The plots indicates that the speedup increases as the number of threads increases.
2. However, the efficiency of the algorithm slightly decreases with an increase in the number of threads.
3. Additionally, the speedups show a slight decrease with increasing problem size.

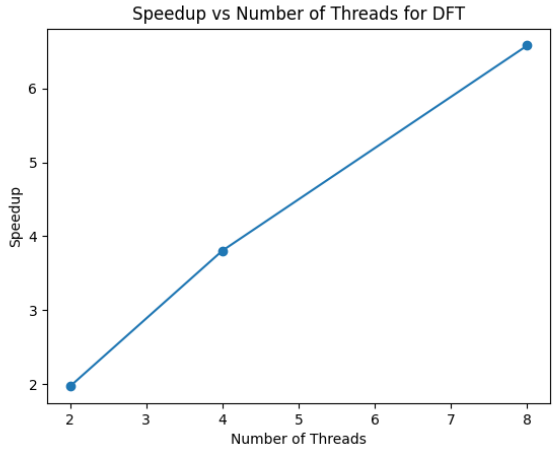

FIGURE 2. OMP DFT SPEEDUPS FOR 355*343 IMAGE

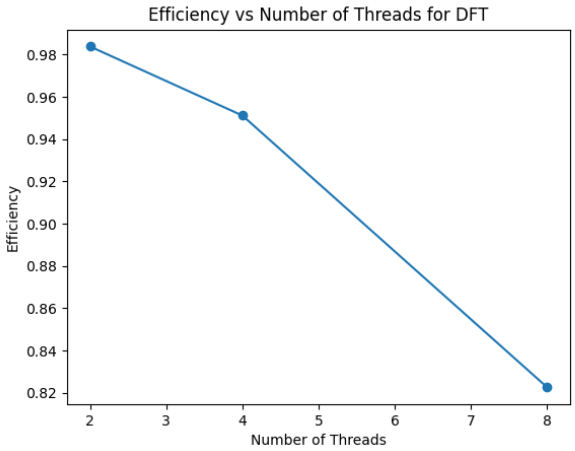

FIGURE 3. OMP DFT EFFICIENCY FOR 355*343 IMAGE

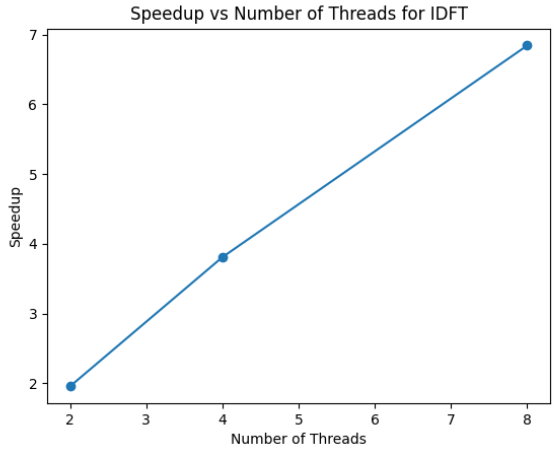

FIGURE 4. OMP IDFT SPEEDUPS FOR 355*343 IMAGE

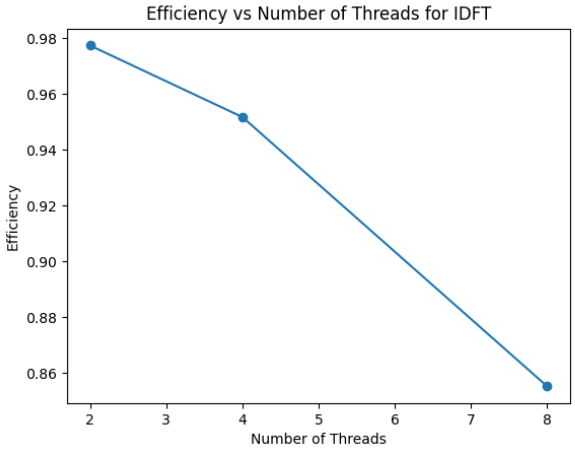

FIGURE 5. OMP IDFT EFFICIENCY FOR 355*343 IMAGE

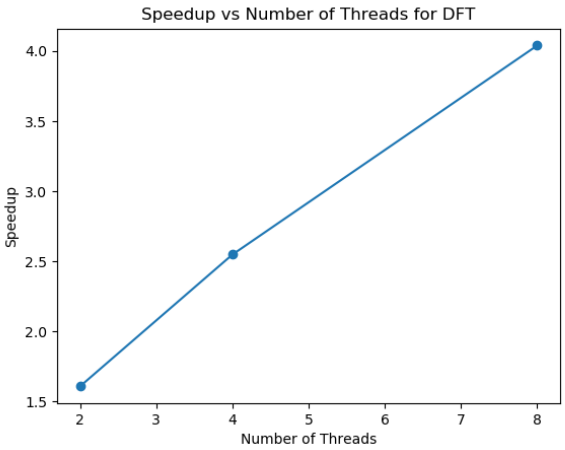

FIGURE 8. MPI DFT SPEEDUPS FOR 64*64 IMAGE

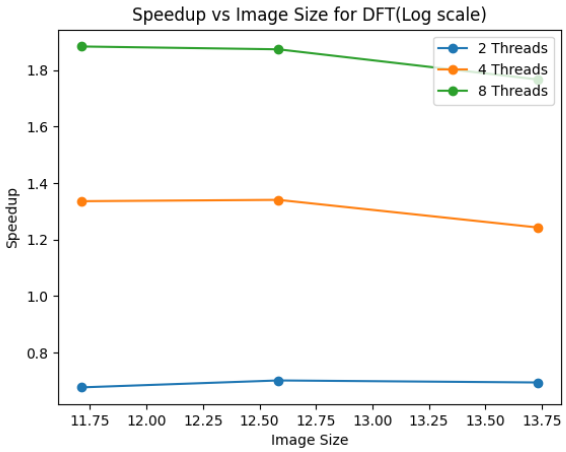

FIGURE 6. OMP DFT SPEEDUPS FOR SCALABILITY

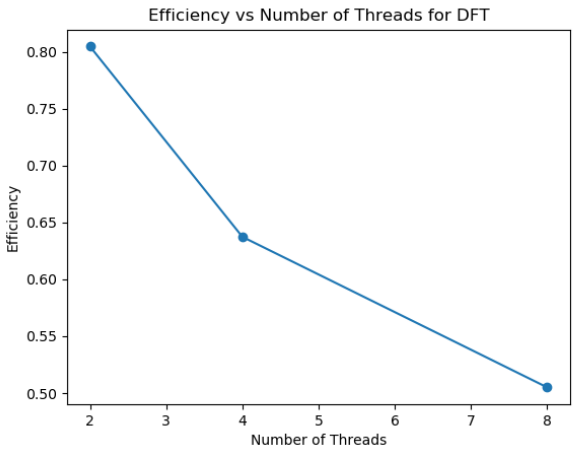

FIGURE 9. MPI DFT EFFICIENCY FOR 64*64 IMAGE

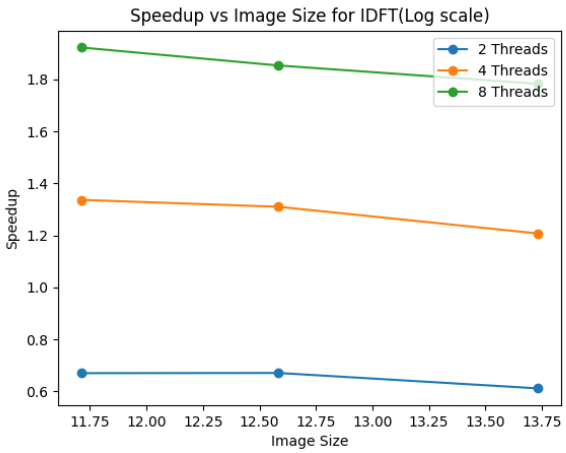

FIGURE 7. OMP IDFT SPEEDUPS FOR SCALABILITY

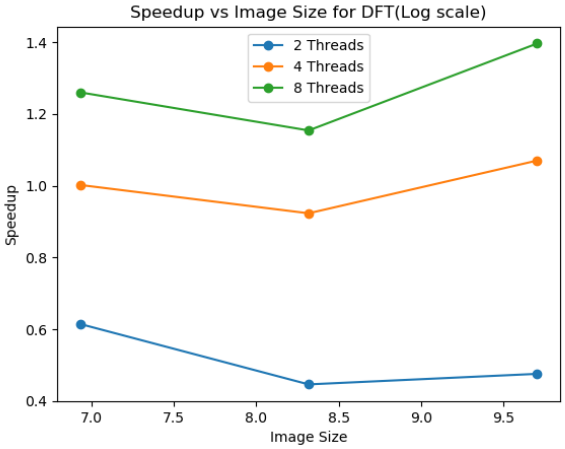

FIGURE 10. MPI DFT SPEEDUPS FOR SCALABILITY

IDFT Results

1. The IDFT algorithm also follows a similar trend to DFT computation,
2. The plots indicates that the speedup increases as the number of threads increases.
3. However, the efficiency of the algorithm slightly decreases with an increase in the number of threads.
4. Additionally, the speedups show a slight decrease with increasing problem size.

OPEN MPI IMPLEMENTATION

For the speedup and efficiency analysis, an image size of 64*64 was utilized. For scalability assessment, image sizes of 32*32, 64*64, and 128*128 were employed. Instead of using an actual image, a Python code was employed to generate the required sized 2D matrix with random values in the range of 0-255.

DFT Results

1. The MPI DFT algorithm also follows a similar trend to OMP DFT computation,
2. The plots indicates that the speedup increases as the number of threads increases.
3. However, the efficiency of the algorithm slightly decreases with an increase in the number of threads.
4. Additionally, the speedups show a slight dip and then increases again with problem size

DISCUSSION

Some applications of Discrete Fourier Transformation are as follows:

1. **Signal Processing**: The DFT is used for filtering, spectral analysis, and compression, allowing signals to be decomposed into their frequency components for analysis and manipulation.
2. **Communications**: In modulation, demodulation, channel equalization, and synchronization, the DFT is essential for processing signals in communication systems.
3. **Image Processing**: For tasks such as image enhancement, compression, and pattern recognition, the DFT is employed to analyze and manipulate images in the frequency domain.
4. **Scientific Computing and Engineering**: The DFT is widely used in solving differential equations, analyzing system dynamics, and simulating physical phenomena in various scientific and engineering applications.
5. **Interdisciplinary Fields**: Beyond traditional domains, the DFT finds applications in astronomy, finance, bioinformatics, and geophysics, reflecting its versatility and importance across diverse fields of study and research.

FAST FOURIER TRANSFORMATION

The Fast Fourier Transform (FFT) is a highly efficient algorithm used to compute the Discrete Fourier Transform (DFT) of a sequence. In contrast to the DFT, which operates in $O(N^2)$ time complexity, the FFT reduces this to $O(N \log N)$, making it notably faster for handling large sequences. The primary distinction between the FFT and the DFT lies in their computational complexity and implementation approach. The FFT achieves its speedup by capitalizing on the symmetry properties of the DFT and recursively dividing the computation into smaller subproblems. This divide-and-conquer strategy significantly reduces the number of operations needed, resulting in faster computation times. However, it is acknowledged that implementing the FFT algorithm can be more challenging compared to the DFT. Therefore, for this project, the DFT was chosen over the FFT due to its simpler implementation.

ACKNOWLEDGEMENT

I would like to express my heartfelt gratitude to Professor Kameswararao Anupindi for his invaluable guidance and support throughout my project on parallelizing Discrete Fourier Transformation using OpenMP and MPI in C++. His expertise, encouragement, and mentorship have been instrumental in shaping my understanding of parallel computing and enhancing my skills in algorithm optimization. I am deeply grateful for his unwavering commitment and insightful feedback, which have significantly contributed to the success of this project.

REFERENCES

Below are the references utilized for this project:

1. Tangirala, A.K. (2015). Principles of System Identification: Theory and Practice (1st ed.). CRC Press. https://doi.org/10.1201/9781315222509
2. Wikipedia article on Discrete Fourier Transformation
3. OpenMP documentation
4. Open MPI documentation

GITHUB REPO

GitHub link to project: Discrete Fourier Transformation