PARALLELIZING FAST FOURIER TRANSFORM

CSE 633 – Aaqib Wadood Syed

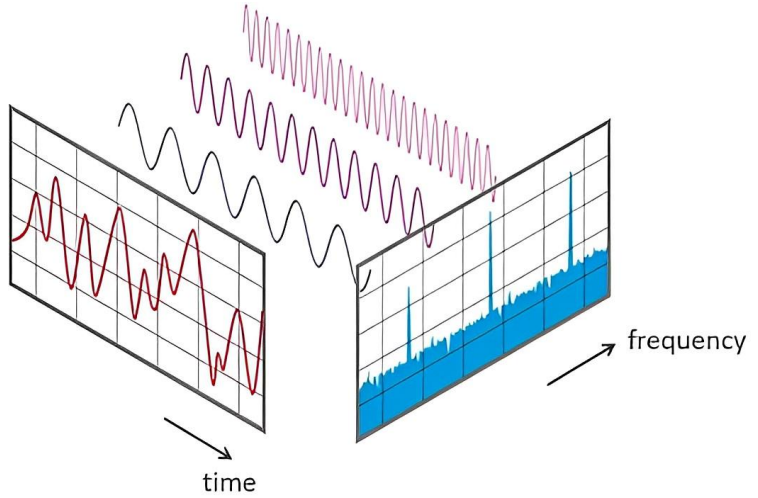
University at Buffalo The State University of New York

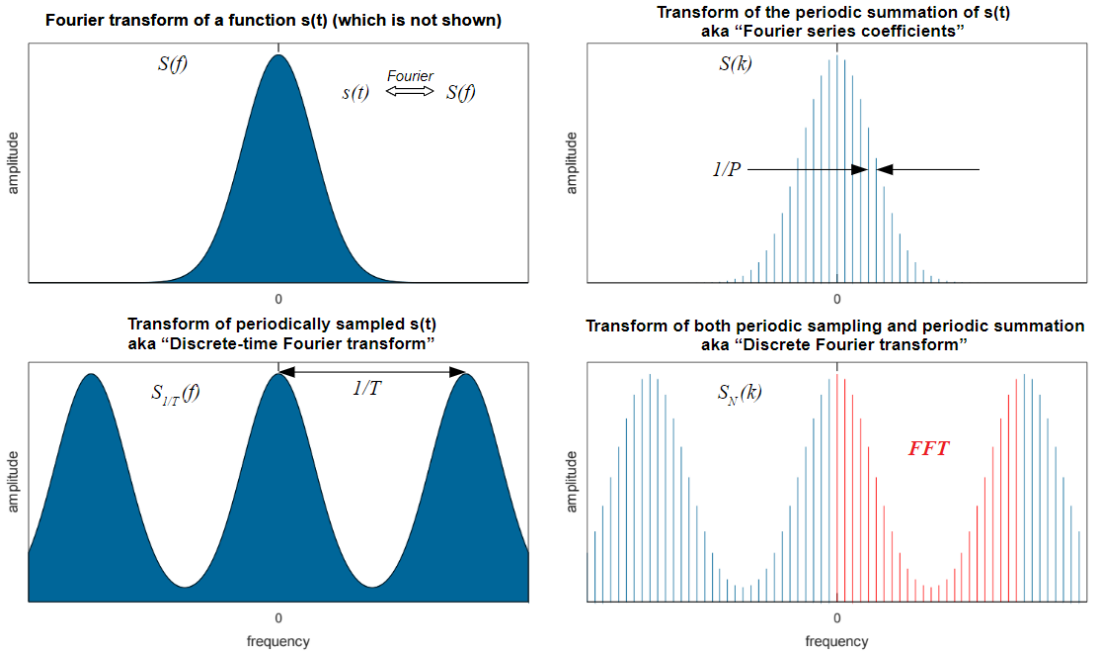
OUTLINE

- Background
- Applications
- Cooley-Tukey Algorithm
- Sequential Implementation
- Parallel Implementation
- Results & Inferences
- To-Do Checklist

BACKGROUND: DEFINING FOURIER TRANSFORM AND DISCRETE FOURIER TRANSFORM

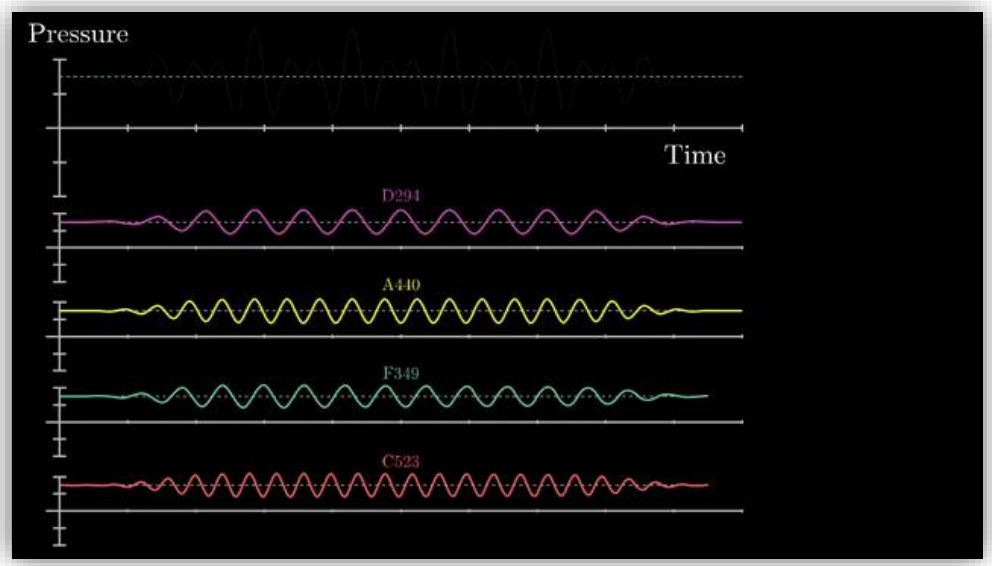
- FT is a mathematical transform that decomposes functions into frequency components, which are represented by the output of the transform as a function of frequency.
- DFT does the same but over a finite sequence of equally-spaced samples of a function.

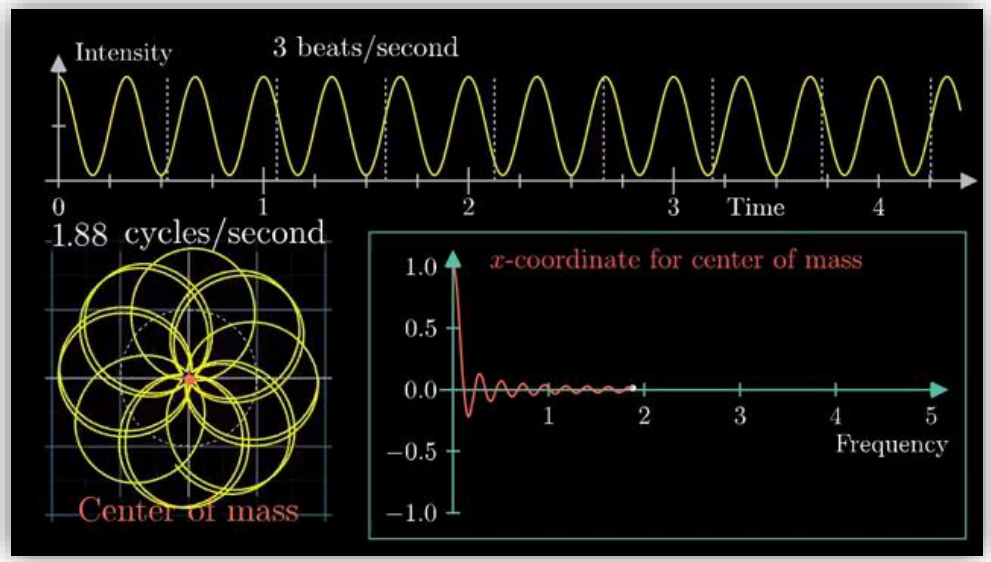




WHAT IS FAST FOURIER TRANSFORM?

- It is an algorithm that takes a discrete time-domain signal and transforms it into a discrete frequency-domain signal efficiently.
- This enables us to identify the frequency components of a signal and analyze its behavior.

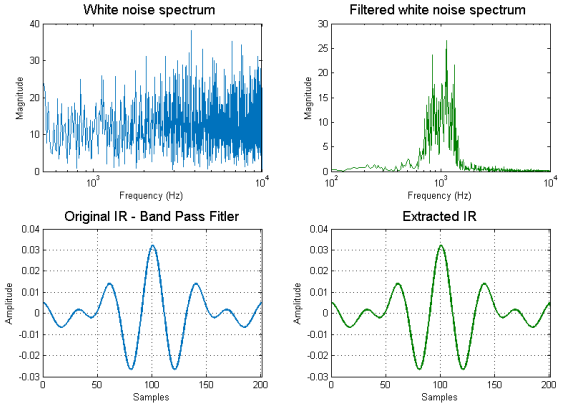




APPLICATIONS

- The FFT (Fast Fourier Transform) algorithm is widely used in diverse fields due to its efficiency in computing the Discrete Fourier Transform. Some of the key applications of the FFT include:

1. Signal processing
2. Audio and image compression
3. Scientific simulations



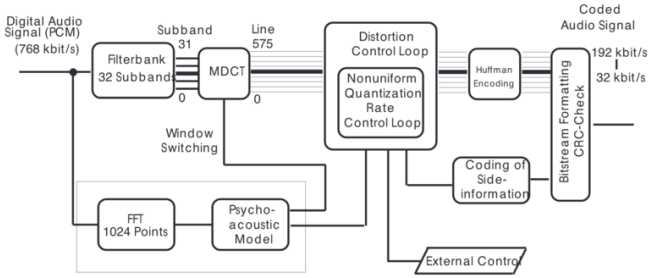
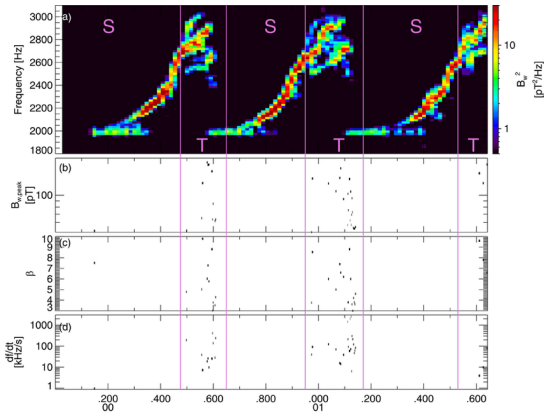
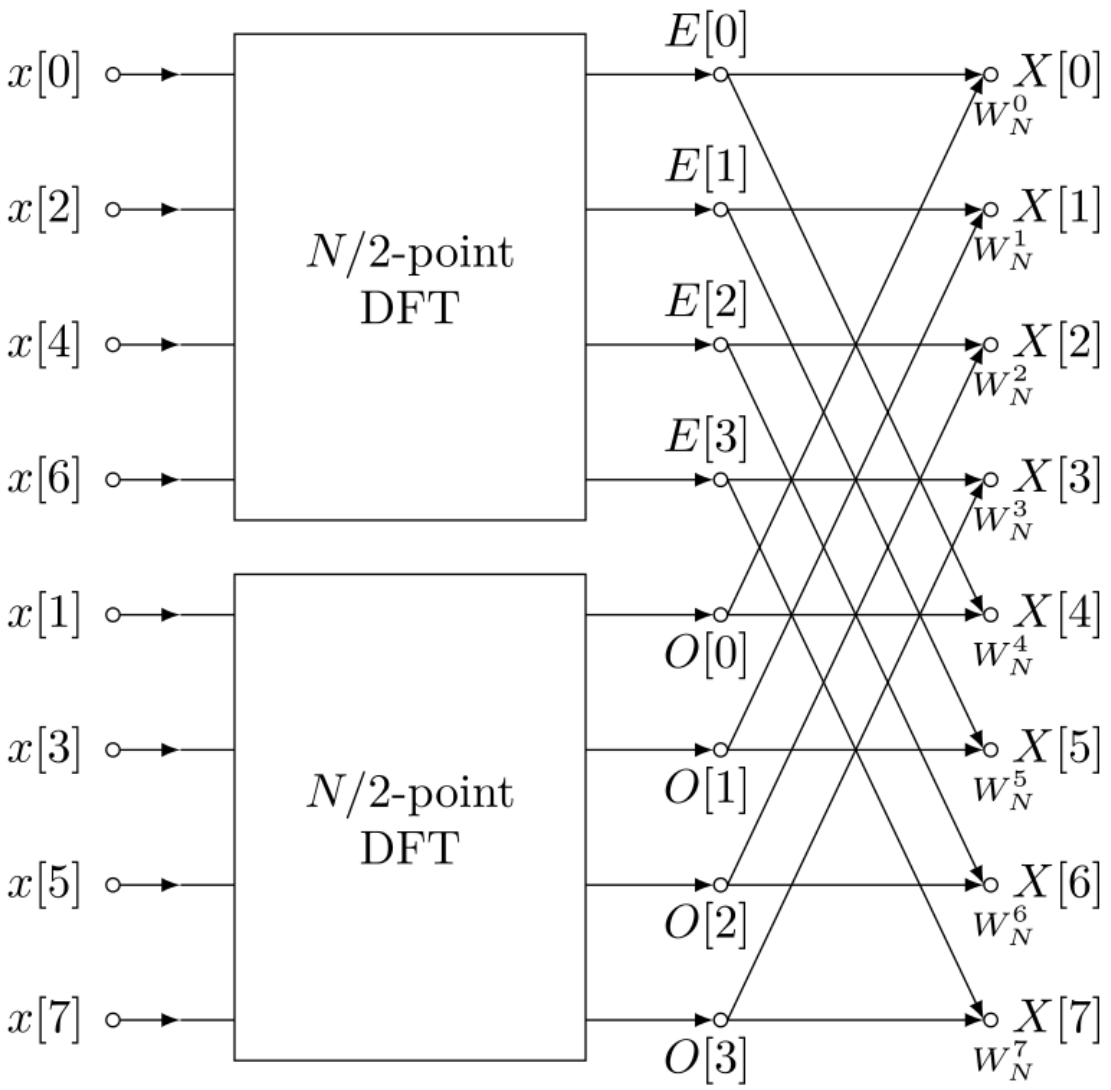


Figure 2: Block diagram of an MPEG-1 Layer-3 encoder.





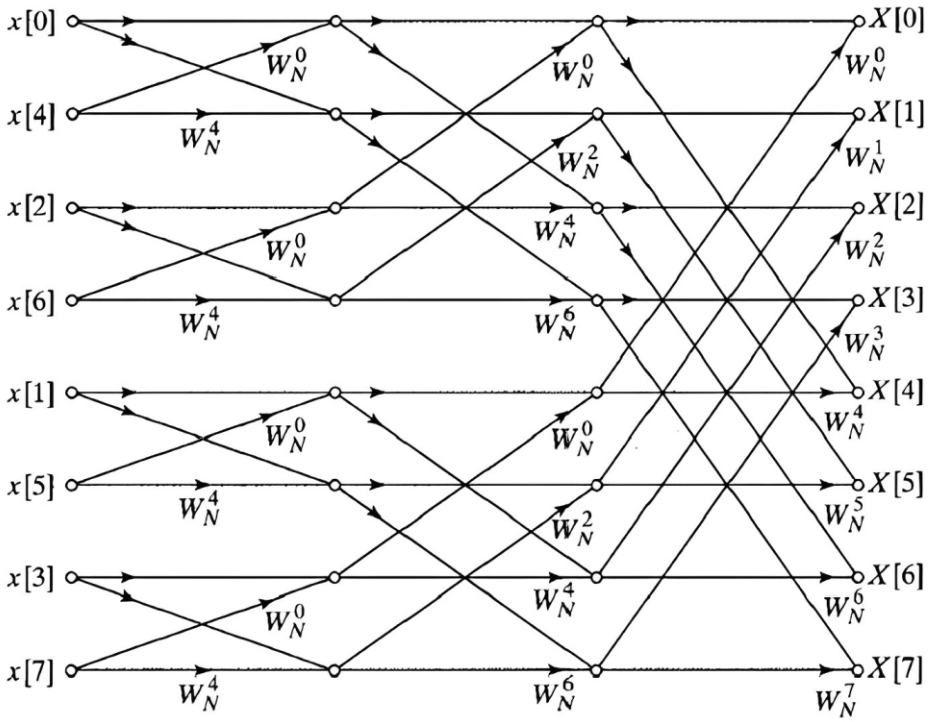
(Source: Wikipedia)

COOLEY–TUKEY ALGORITHM

- Simplest and most common approach is the “radix-2” Decimation-In-Time approach.
- The input sequence is repeatedly divided into two smaller even and odd sub-sequences, and their DFTs are combined using a butterfly operation.

  Strategy: Divide-and-Conquer

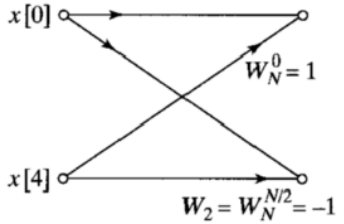
  Complexity: O(NlogN)



(Source: Wikipedia)

Twiddle factor:

$W_N^k = e^{-i2\pi k/N}$

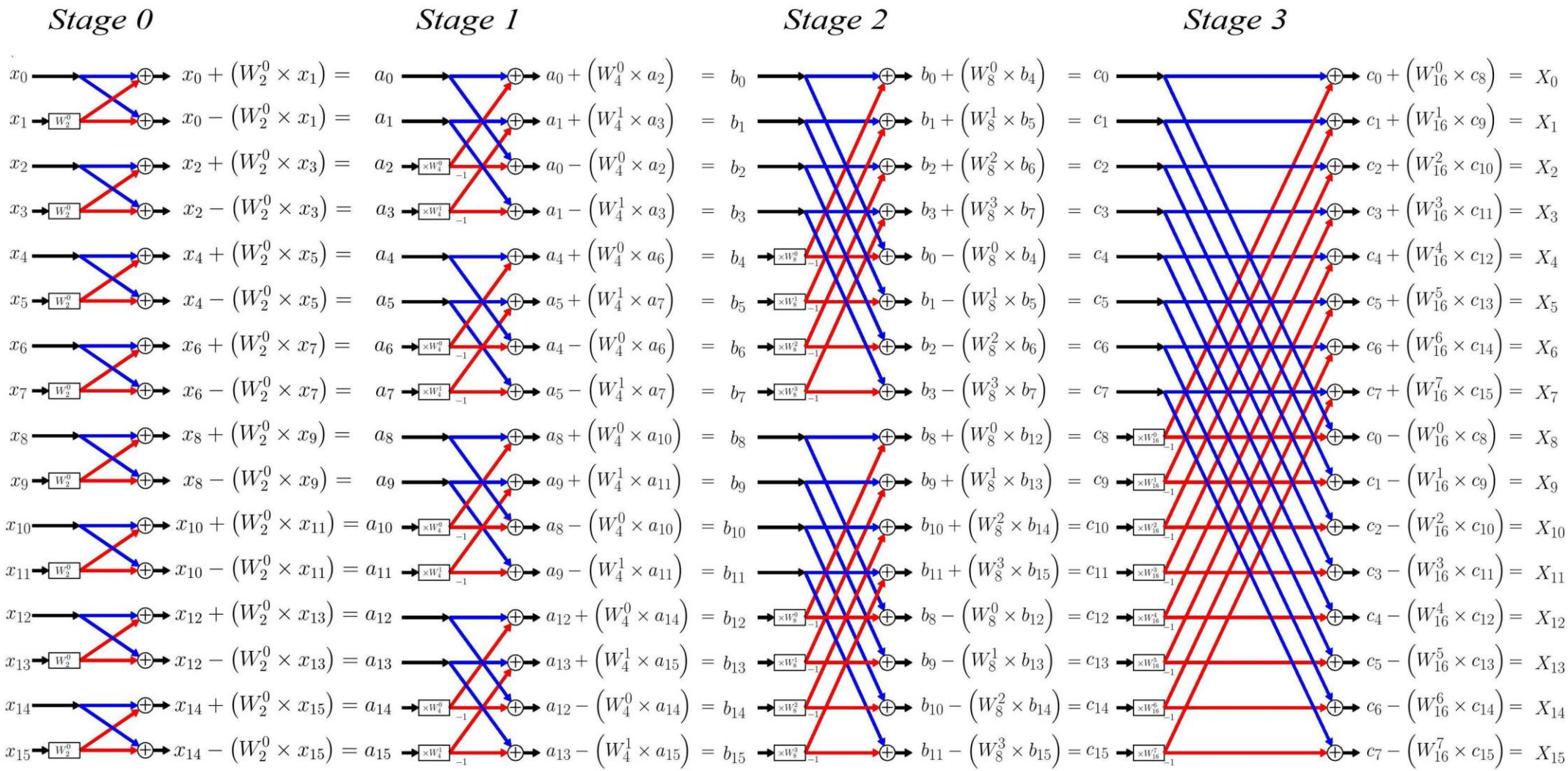


(Basic Butterfly Unit)

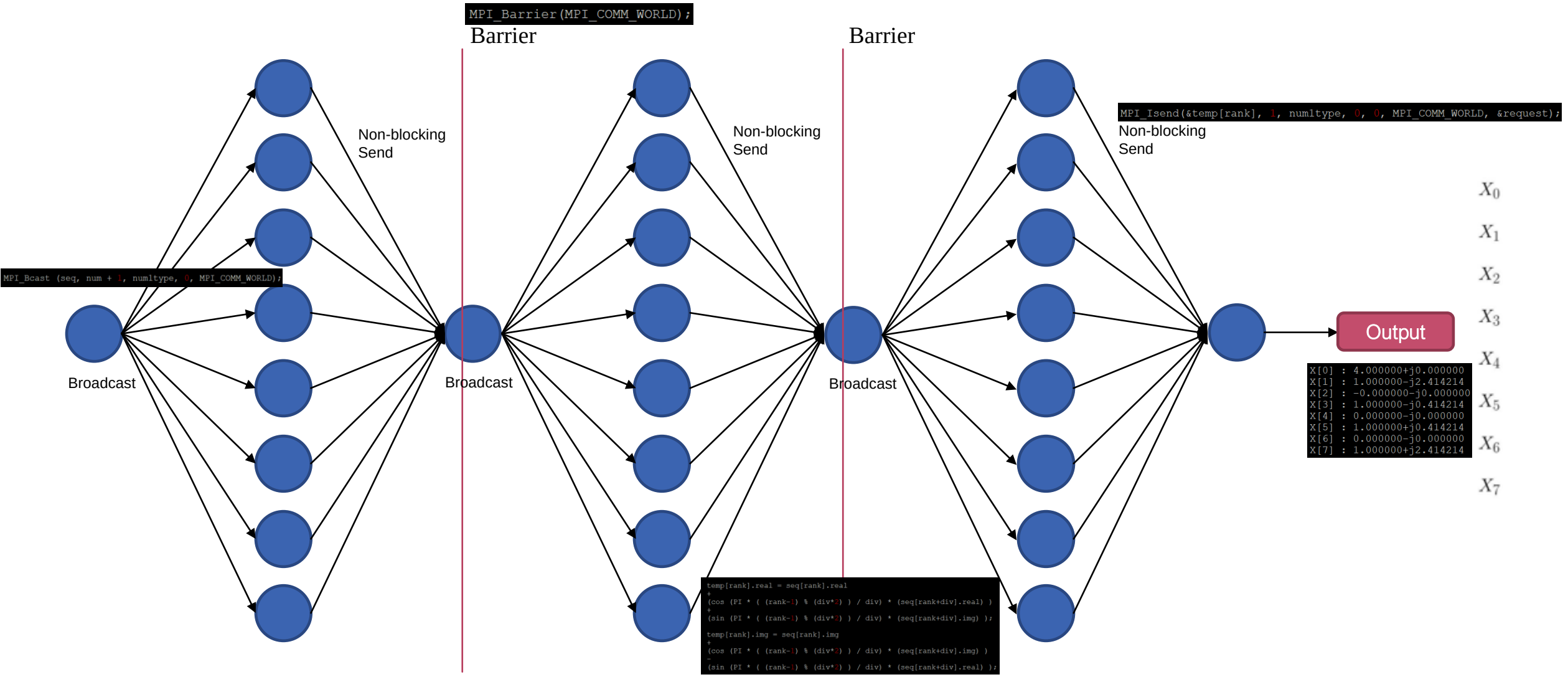
PARALLEL IMPLEMENTATION

- The Fast Fourier Transform (FFT) algorithm utilizes a Butterfly Topology as its fundamental building block, which can be efficiently parallelized to improve computation speed and reduce processing time.

SEQUENTIAL VS PARALLEL



PARALLEL IMPLEMENTATION USING MPI



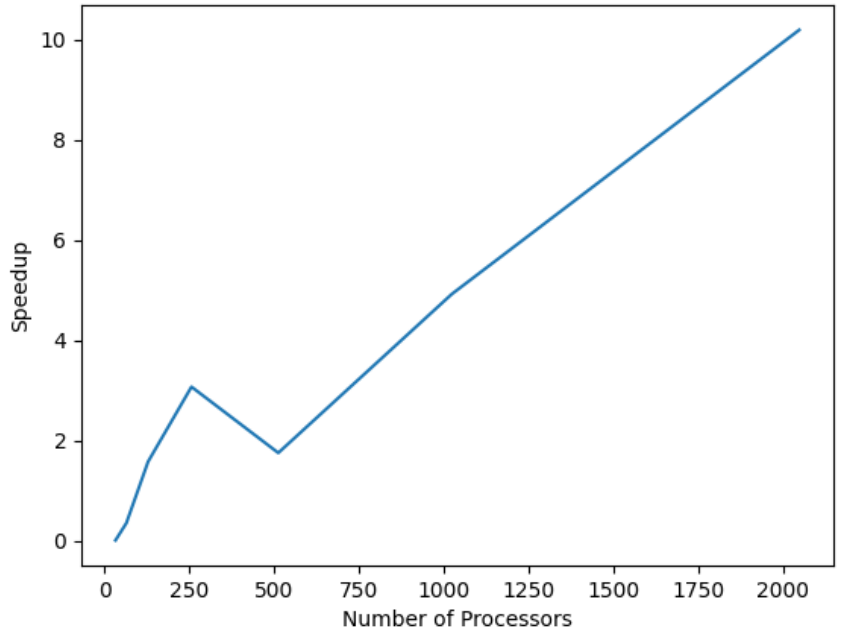
SLURM SCRIPT

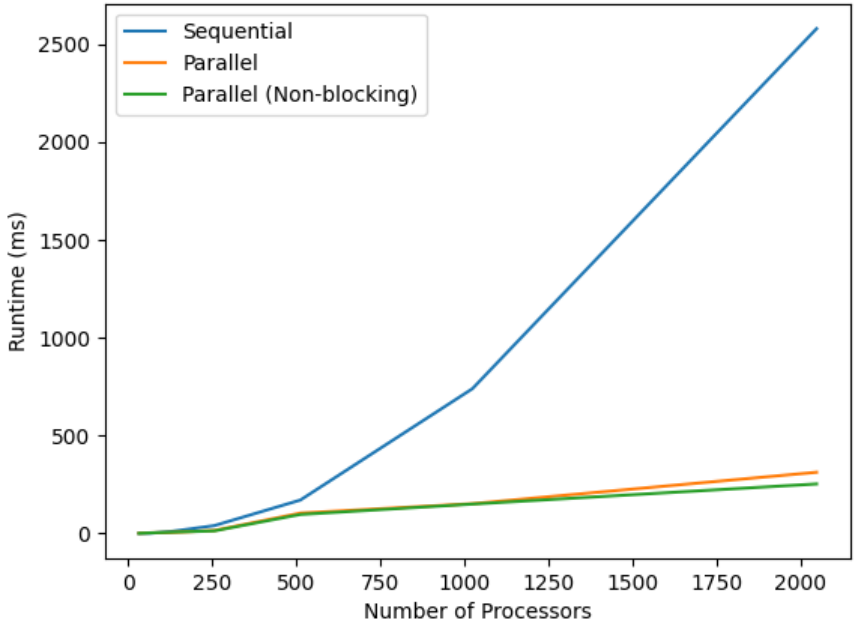
aaqibwad@srv-p22-12:~

```
#!/bin/sh
#SBATCH --partition=general-compute
#SBATCH --qos=general-compute
#SBATCH --time=00:05:00
#SBATCH --nodes=128
#SBATCH --ntasks-per-node=16
#SBATCH --mem=50000
#SBATCH  --output=fft.out
#SBATCH --job-name="parallel_fft"
module load intel/17.0
module load intel-mpi/2017.0.1
export I_MPI_PMI_LIBRARY=/usr/lib64/libpmi.so
mpicc -o fft_para fft_parallel.c -lm
srun ./fft_para
~
```

RESULTS

# of Processors (# of Inputs)	Sequential Execution Time (ms)	Parallel Execution Time (ms)	Parallel Execution Time (ms) (Non-blocking)
32	0.01	1	0.7
64	1	3	2.8
128	10	5.1	6.32
256	40	15	13
512	170.61	104	97
1024	740	152	150.1
2048	2580.45	312.75	253

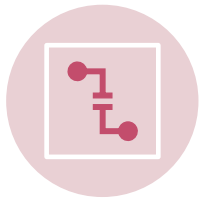




INFERENCES

- As the number of processors increases, the speedup improves.
- Non-blocking I/O reduces communication overhead and improves scalability of parallel algorithms.
- When using MPI, processors on the same node utilize shared memory implicitly instead of message passing for communication leading to more efficient communication.

TO-DO CHECKLIST



Try to establish direct communication between worker nodes for data passing.

Evaluate performance for a higher number of inputs.

Explore non-blocking communication functions like MPI_Isend() and MPI_Irecv().

REFERENCES

- Chu, E., & George, A. (1990). FFT Algorithms and their adaptation to parallel processing. IEEE Transactions on Computers, 39(3), 420-430.
- Halawi, O. N. (2010). Parallel Fast Fourier Transform. Journal of Parallel and Distributed Computing, 70(1), 1-14.
- "Fourier Transform - A visual introduction" - 3Blue1Brown [Youtube]
- "The Remarkable Story Behind The Most Important Algorithm Of All Time“ -Veritasium [Youtube]


QUESTIONS