1D FFT: Computational Implementation Using Matrices

The **Fast Fourier Transform (FFT)** is an efficient algorithm for computing the **Discrete Fourier Transform (DFT)** using matrix operations. Let's explore its **matrix representation** and computational aspects relevant to assembly implementation.

---

1. Matrix Representation of DFT

The **DFT** of a signal x of length N can be written as a **matrix-vector multiplication**:

$$X = W_N \cdot x$$

where:

- X is the **DFT output** vector ($N \times 1$),
- x is the **input signal** vector ($N \times 1$),
- W_N is the **DFT matrix** ($N \times N$), defined as:

$$W_N = \begin{bmatrix} W_N^{0\cdot 0} & W_N^{0\cdot 1} & W_N^{0\cdot 2} & \dots & W_N^{0\cdot (N-1)} \\ W_N^{1\cdot 0} & W_N^{1\cdot 1} & W_N^{1\cdot 2} & \dots & W_N^{1\cdot (N-1)} \\ W_N^{2\cdot 0} & W_N^{2\cdot 1} & W_N^{2\cdot 2} & \dots & W_N^{2\cdot (N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ W_N^{(N-1)\cdot 0} & W_N^{(N-1)\cdot 1} & W_N^{(N-1)\cdot 2} & \dots & W_N^{(N-1)\cdot (N-1)} \end{bmatrix}$$

where the twiddle factors WNkW_N^k are given by:

$$W_N = e^{-j\frac{2\pi}{N}}$$

Each element of the matrix is:

$$W_N^{mn} = e^{-j\frac{2\pi}{N}mn} = \cos\left(\frac{2\pi}{N}mn\right) - j\sin\left(\frac{2\pi}{N}mn\right)$$

This **direct matrix multiplication** requires $O(N^2)$ operations, which is inefficient for large NN. FFT reduces this to $O(N \log N)$ by exploiting the recursive structure.

---

2. FFT as a Recursive Matrix Factorization

Instead of computing the full DFT matrix multiplication, FFT **factorizes** W_N into smaller matrices using a divide-and-conquer approach.

Radix-2 Decimation-In-Time (DIT) FFT

For $N = 2^m$, we split x into even and odd indexed components:

$$x_{even} = \begin{bmatrix} x[0] \\ x[2] \\ x[4] \\ \vdots \end{bmatrix}$$

$$x_{odd} = \begin{bmatrix} x[1] \\ x[3] \\ x[5] \\ \vdots \end{bmatrix}$$

Using this, we can write:

$$X[k] = X_{\text{even}}[k] + W_N^k + X_{\text{odd}}[k]$$
$$X[k + N/2] = X_{\text{even}}[k] - W_N^k X_{\text{odd}}[k]$$

Where $W_N^k = e^{-j\frac{2\pi}{N}k}$ is the **twiddle factor**.

This recursive structure continues until we reach **2-point DFTs**:

$$X[0] = x[0] + x[1]$$
$$X[1] = x[0] - x[1]$$

Thus, an **N-point FFT is computed using log_2(N) stages**, each involving **N/2 butterfly operations**.

---

3. Matrix Form of FFT (Block Factorization Approach)

Instead of computing the full W_N matrix, we **factorize it recursively** into block diagonal matrices.

For N = 4:

$$W_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W_4^1 & W_4^2 & W_4^3 \\ 1 & W_4^2 & W_4^4 & W_4^6 \\ 1 & W_4^3 & W_4^6 & W_4^9 \end{bmatrix}$$

Using **recursive factorization**:

$$W_4 = \begin{bmatrix} I & D_2 \\ I & -D_2 \end{bmatrix} \cdot \begin{bmatrix} W_2 & 0 \\ 0 & W_2 \end{bmatrix}$$

where:

- II is the identity matrix.
- D_2 is the diagonal matrix of twiddle factors.
- W_2 represents the DFT matrix of size 2×2.

For **general N (power of 2)**:

$$W_N = \begin{bmatrix} I & D_{N/2} \\ I & -D_{N/2} \end{bmatrix} \cdot \begin{bmatrix} W_{N/2} & 0 \\ 0 & W_{N/2} \end{bmatrix}$$

This recursive matrix formulation leads to efficient **vectorized butterfly operations**, ideal for RISC-V vector assembly.

---

4. Computational Implementation in Assembly

Using this matrix-based approach, an **efficient assembly implementation** will involve:

1. **Bit-Reversal Permutation** (Rearrange indices before computation)
    - Use a lookup table for bit-reversed indices.
    - Store inputs accordingly.
2. **Butterfly Computation** (Core FFT Step)
    - Load **two complex values** (even and odd indices).
    - Compute sum and difference.
    - Multiply the difference by the **twiddle factor** (vectorized multiply-add).
    - Store the result.
3. **Twiddle Factor Computation**
    - Store **precomputed cosine/sine values** in a table.
    - Use **SIMD vector multiplication**.

4. **Recursive Processing**
    - Apply **log_2(N) stages** of butterfly updates.

---

5. Example Input for a 1024-Point FFT

A **suitable input signal** for testing a **1024-point FFT**:

Cosine Wave (Single Tone)

$$x[n] = \cos\left(\frac{2\pi fn}{1024}\right)$$

where f= 100 Hz.

Complex Input (Simulating Real-World Data)

Use both real and imaginary parts:

$$x[n] = \cos\left(\frac{2\pi fn}{1024}\right) + j\sin\left(\frac{2\pi fn}{1024}\right)$$

This tests FFT's ability to handle complex-valued inputs.

Conclusion

- The FFT computation **factorizes the DFT matrix** using recursive block matrices.
- The **butterfly operation** is the core computation and should be **vectorized** in assembly.
- **Bit-reversal indexing, twiddle factor multiplication, and butterfly updates** must be implemented efficiently.
- **SIMD-based vector multiplication** in RISC-V assembly speeds up FFT computation.